



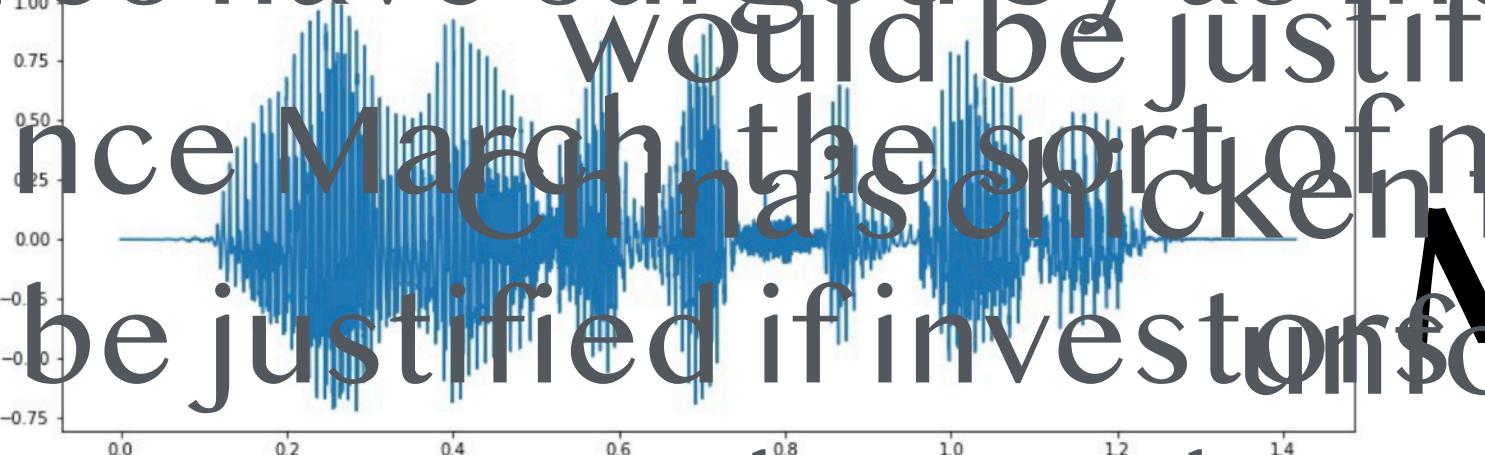
IT UNIVERSITY OF COPENHAGEN

# Lecture 2: Encoder-Decoder Models

Barbara Plank  
LMU Munich

AthNLP2024  
Athens, Greece





# Motivation

Egg futures have surged by as much as one-third since March, the sort of move that would be justified if investors believed China's chicken flocks were headed for an understanding supply and demand. Motivation is what makes people dislike another stock. This comes down

# China's chicken market's dead for anti-company ads

(most?) interesting data is sequential in nature

However, there are 11 Republican members of the House who face competitive races according to Cook. The only one to have endorsed Trump is Tom Reed, the incumbent from New York's 23rd district.

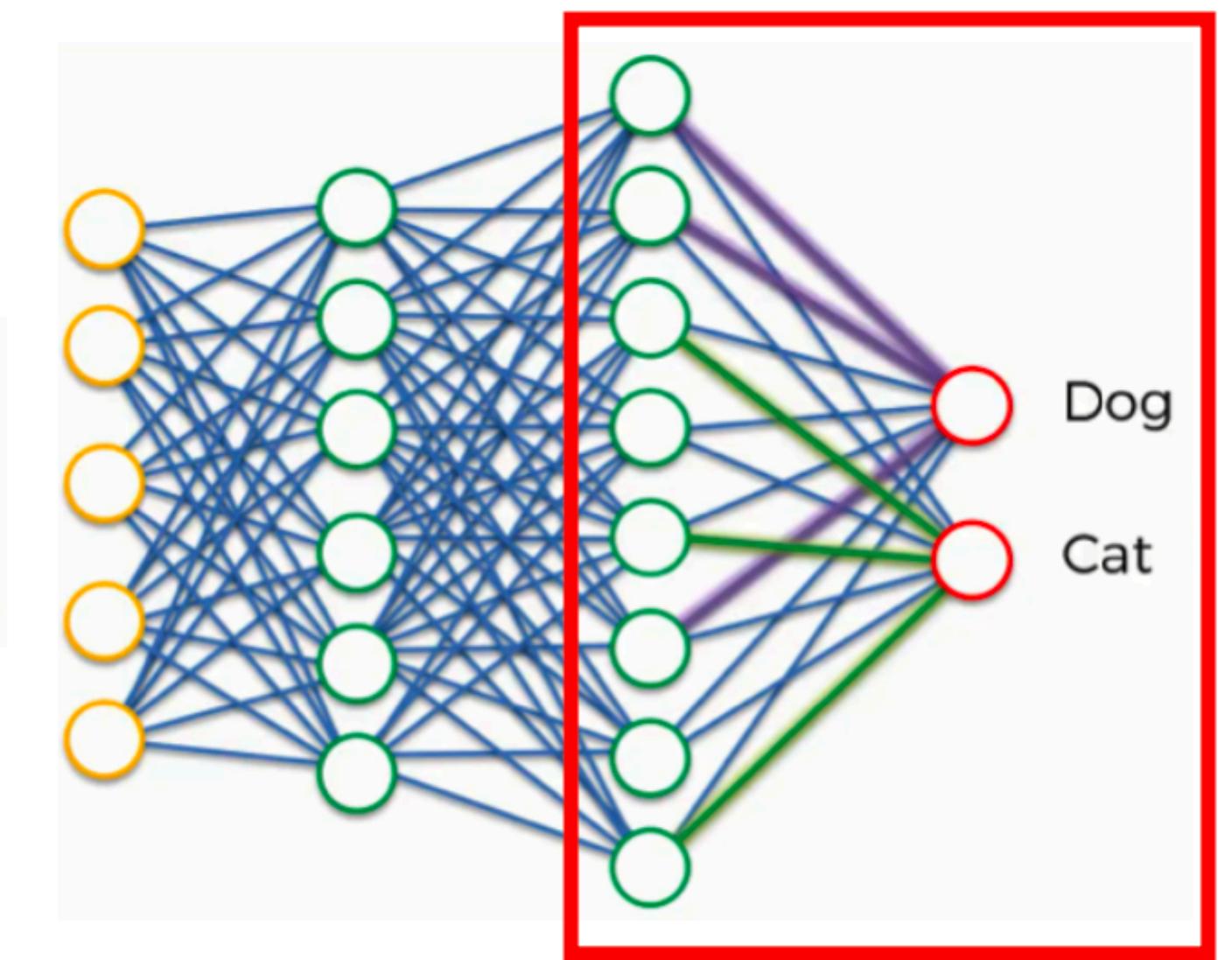
# New York's 23rd Congressional District, a Political Goldmine

# Republican-leaning swing district that covers much of the rural, western part of the state

# What you have heard so far:

- ▶ **Lecture 1: ML Fundamentals**
  - ▶ Linear Classifiers
  - ▶ Non-Linear Classifiers
  - ▶ Feedforward Neural Network
  - ▶ CNN
  - ▶ Feature Representations
  - ▶ Sparse binary features
  - ▶ Continuous dense features  
(word2vec, embedding layers)

## Linear Classifiers and Neural Networks

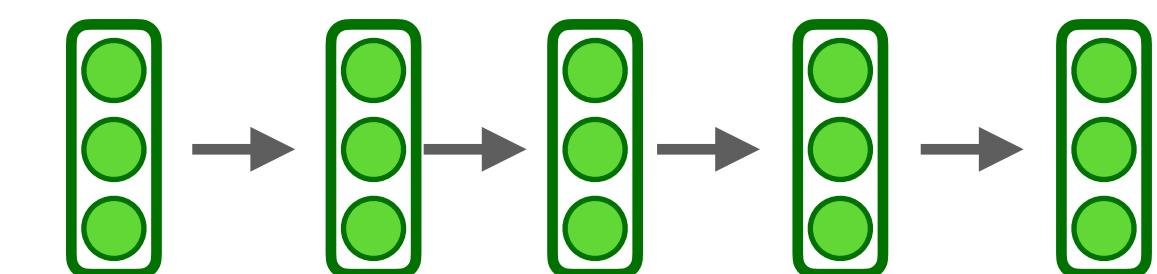


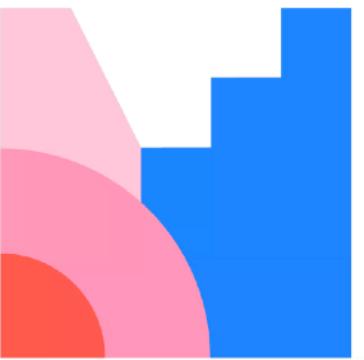
Linear Classifier

# How to deal with sequences?

# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution





# Before we start

[www.menti.com](https://www.menti.com) - Code 2727 6277

# Fundamentals

Part I

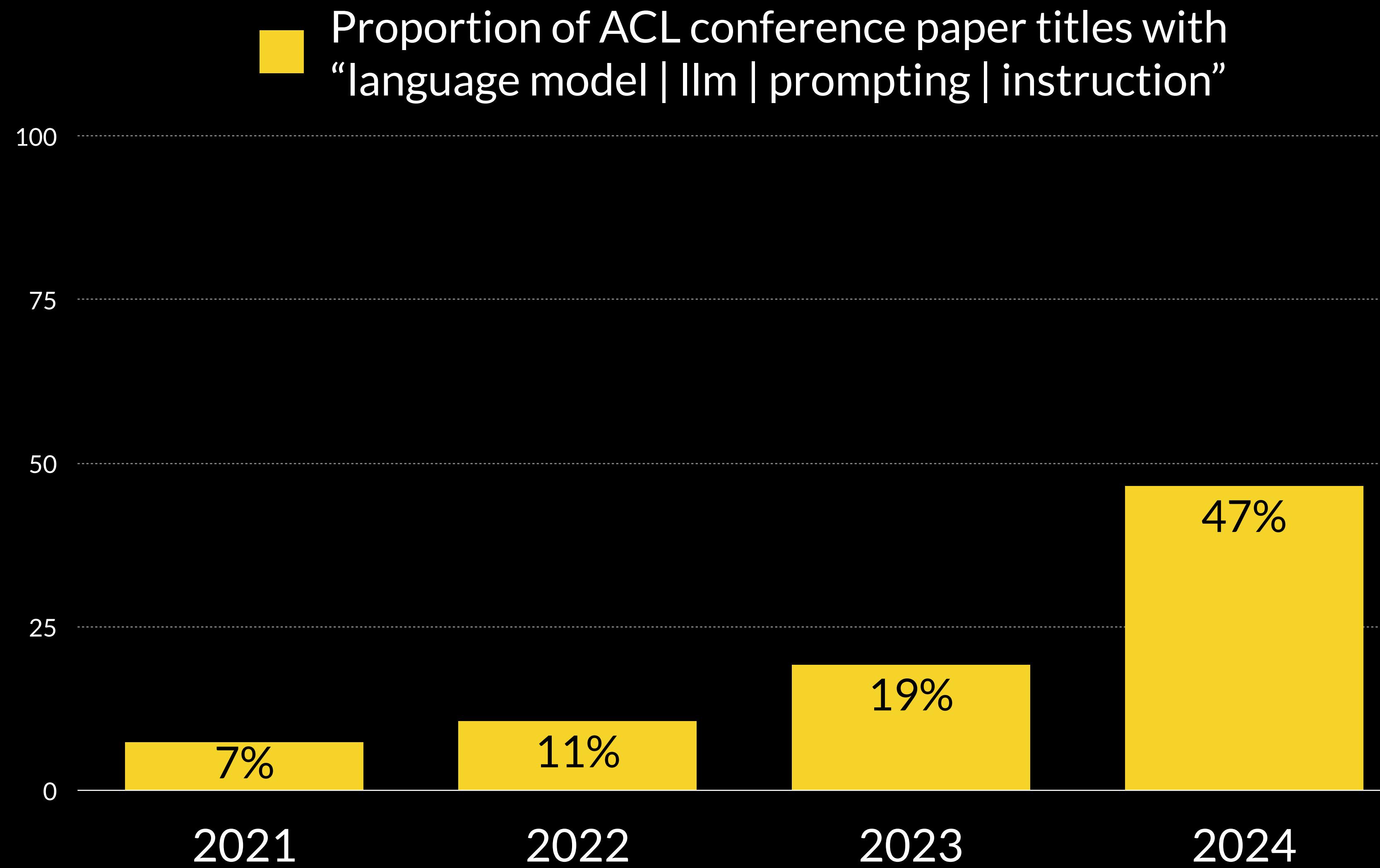
# NLP today: LLMs everywhere!

---



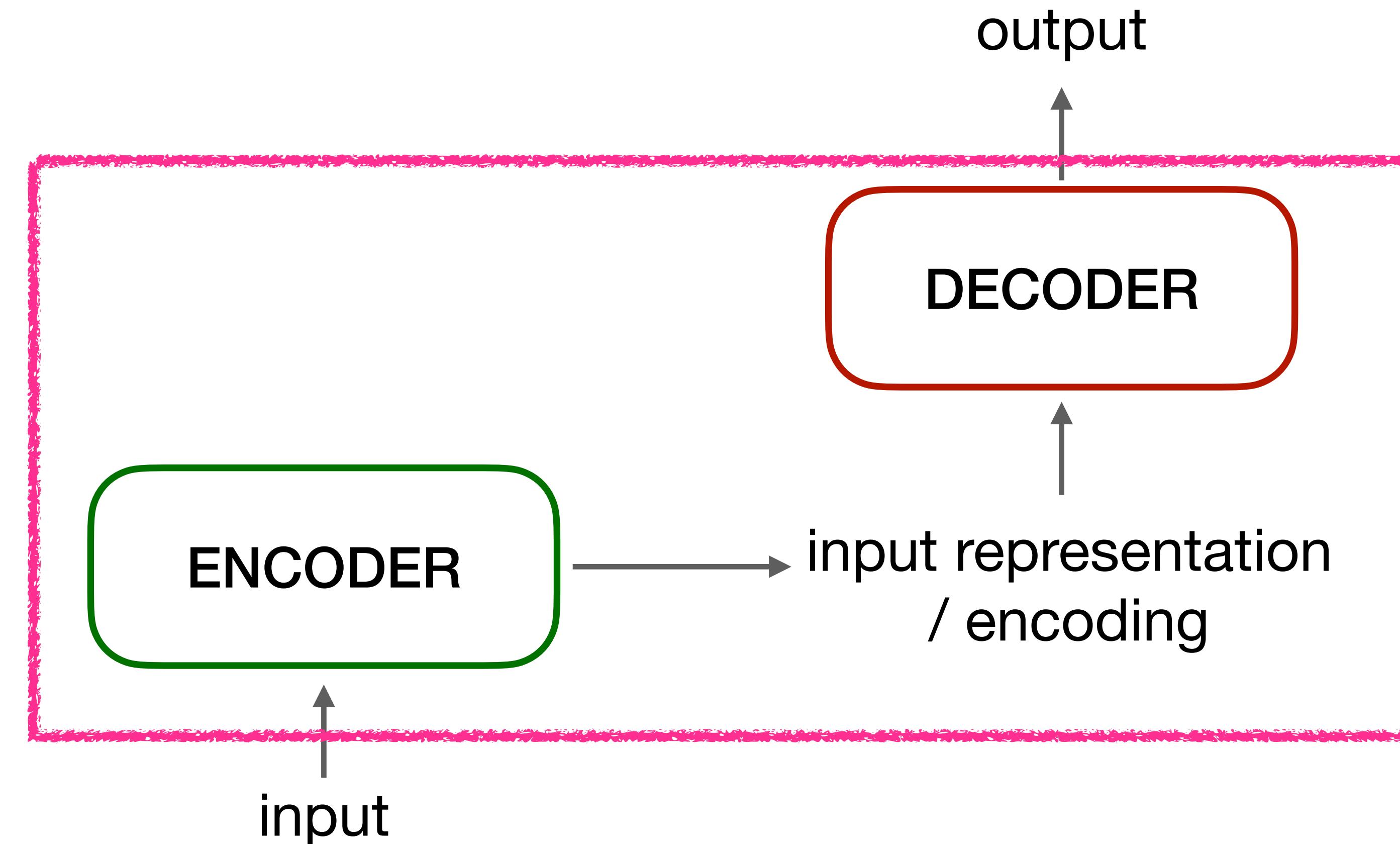
# LLMs: In Recent NLP Research

---



# Encoder-decoder models

general family of models

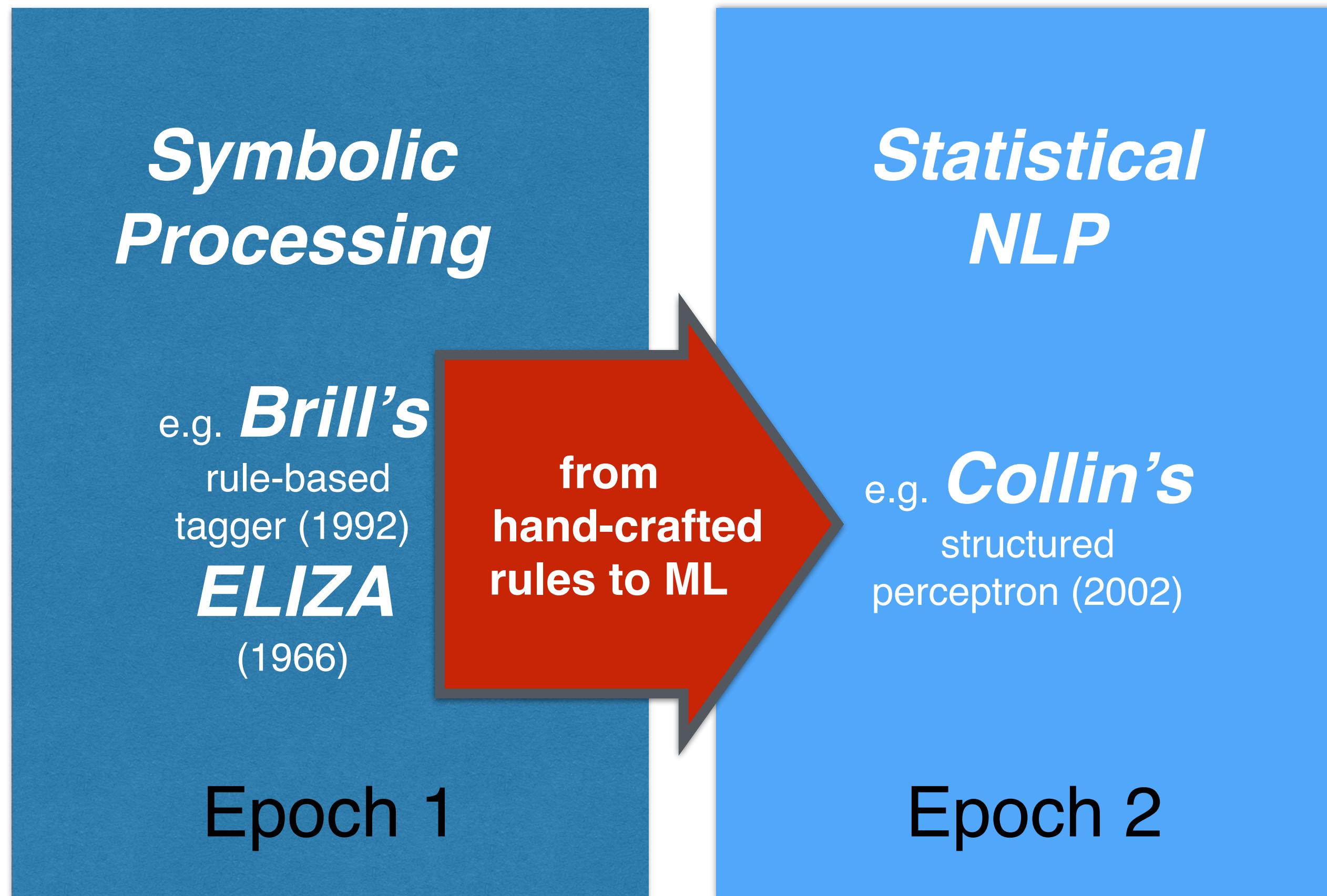


A step back...  
How did the field evolve?

# NLP

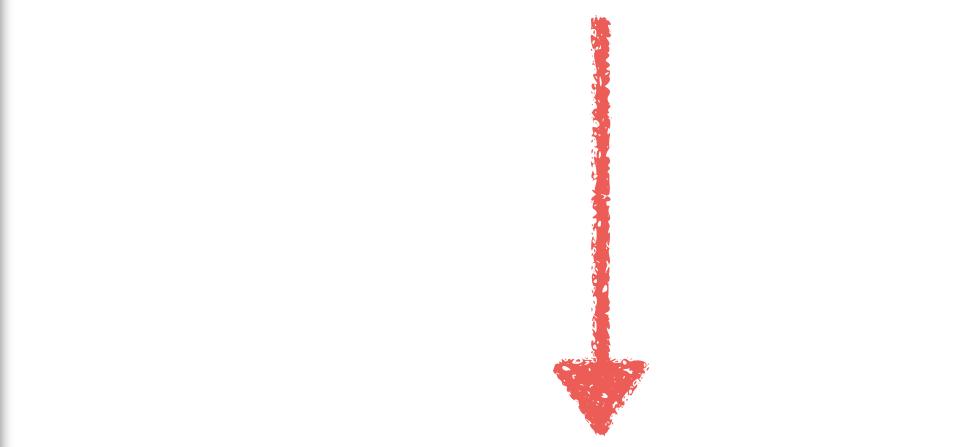


# Machine Learning



approx. 1980s

**x**: the dog barks  
if prev\_w = DET and ...  
tag=NOUN



**x**: 

1	0	..	0	1	0
---	---	----	---	---	---

  
↑                      ↑  
w<sub>i</sub>=dog    w<sub>i-1</sub>=the

classic **sparse** n-hot encodings

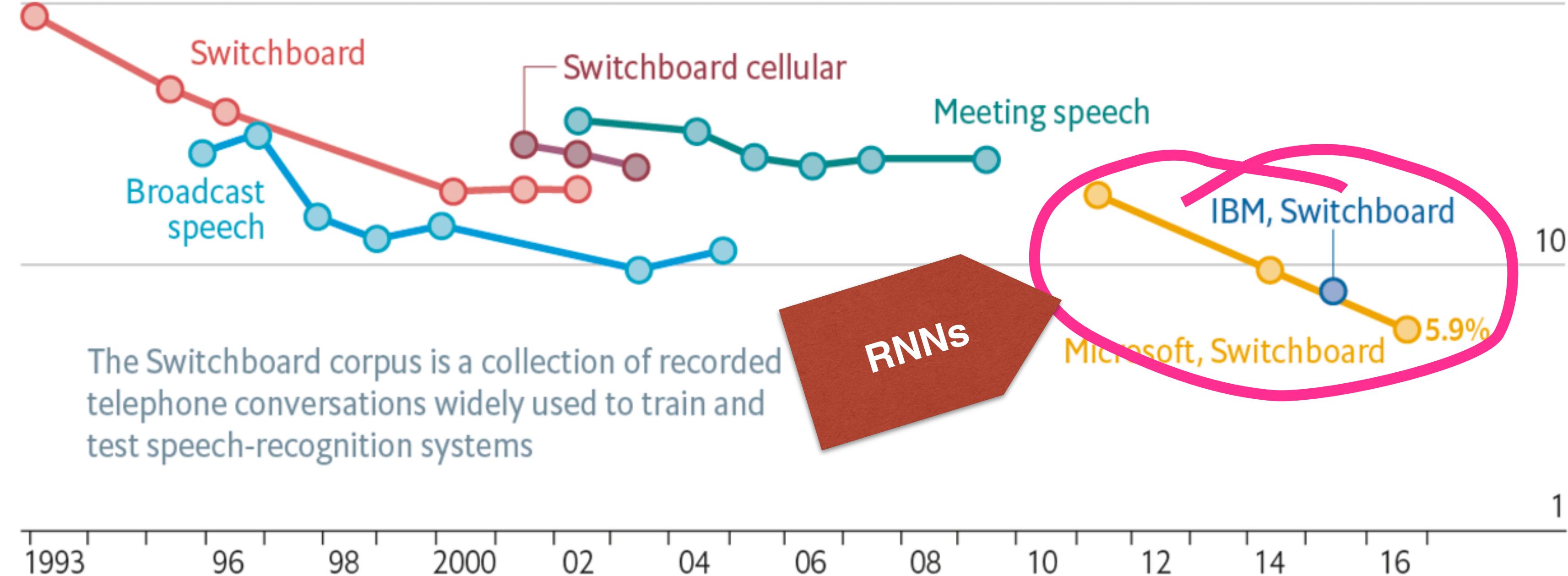
# The emergence of deep learning (in NLP)

# In Speech Recognition

## Loud and clear

Speech-recognition word-error rate, selected benchmarks, %

Log scale  
100



The Switchboard corpus is a collection of recorded telephone conversations widely used to train and test speech-recognition systems

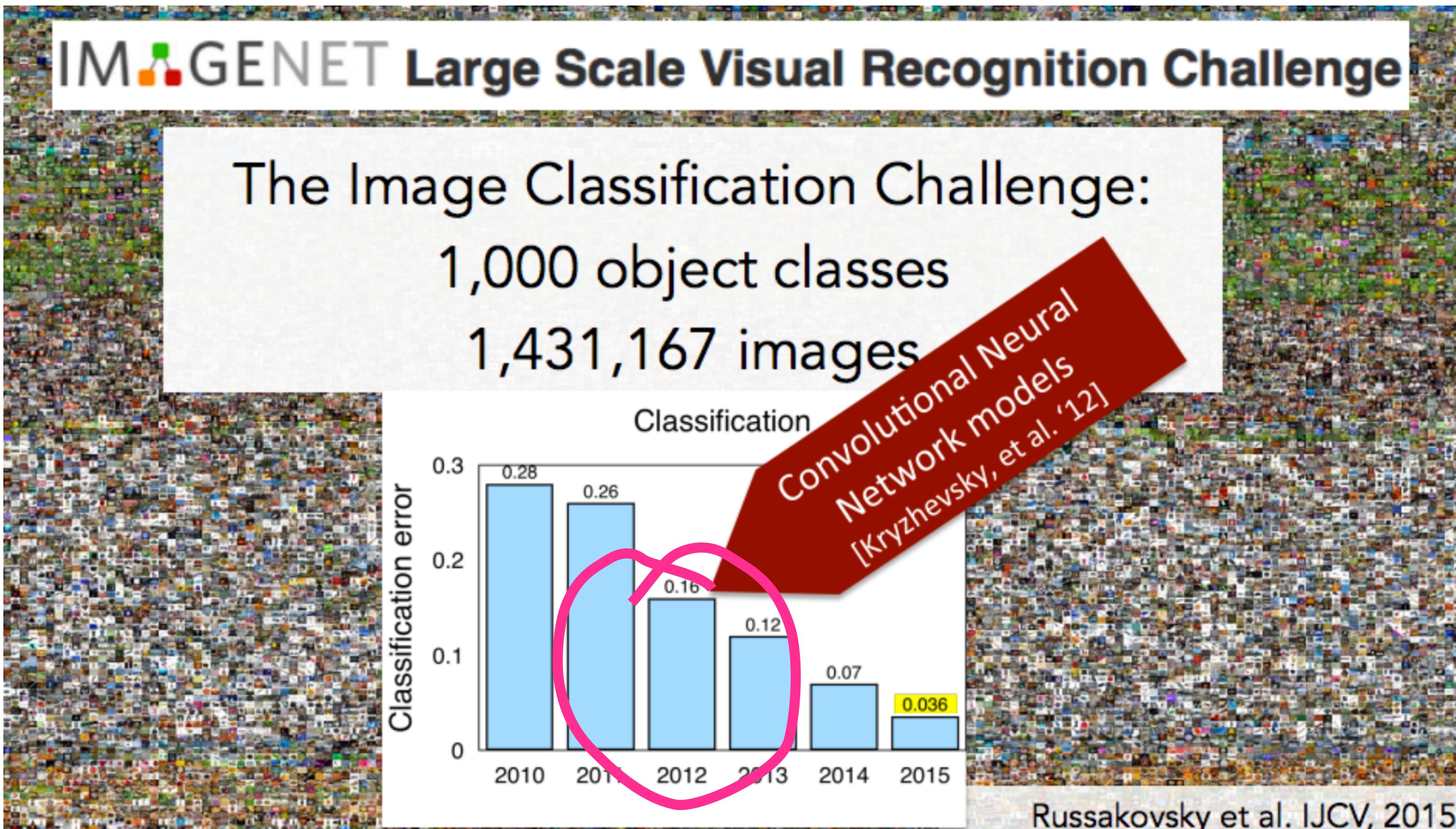
Sources: Microsoft; research papers

(Source: The Economist)

2010



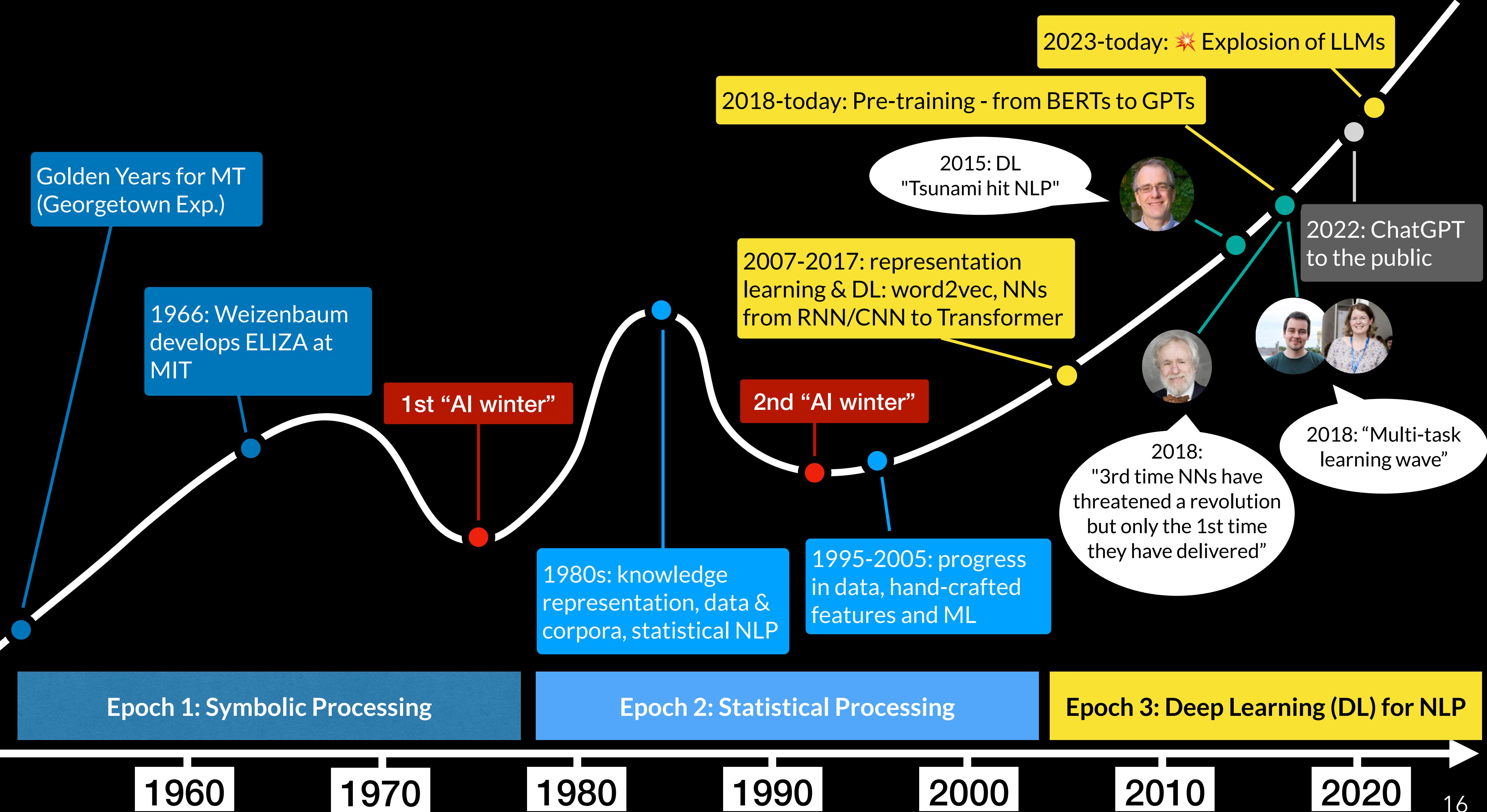
# In Computer Vision



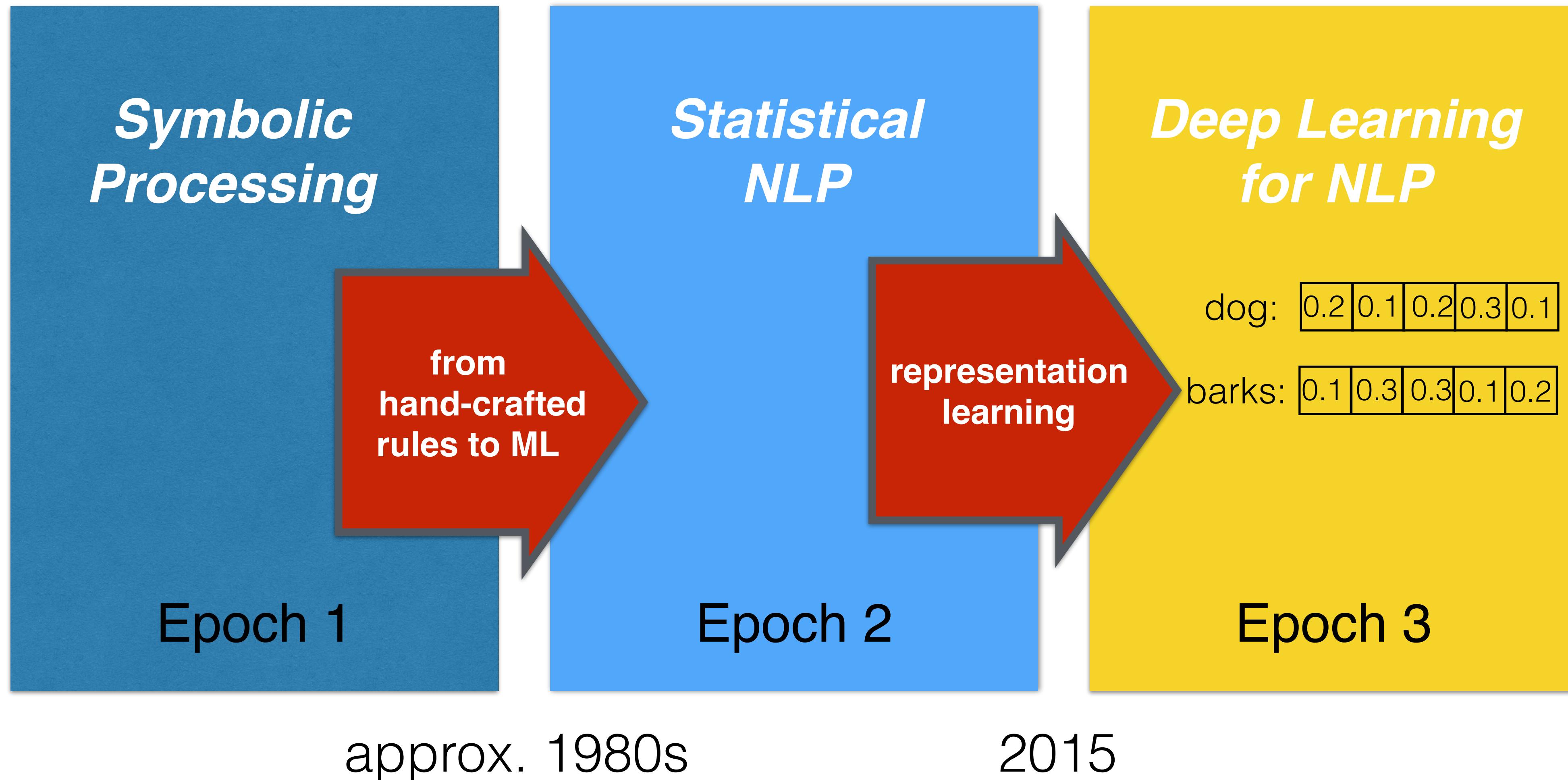
(src: slide by Fei-Fei Li)

2012





# NLP ❤️ Deep Learning



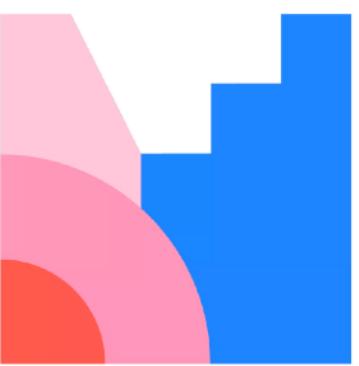
**Back to the roots:  
(traditional)  
Language Models (LMs)**

# Predicting the next word: A Simple (?) Exercise

- ▶ [www.mentimeter.com](http://www.mentimeter.com)  
Room: (see code)

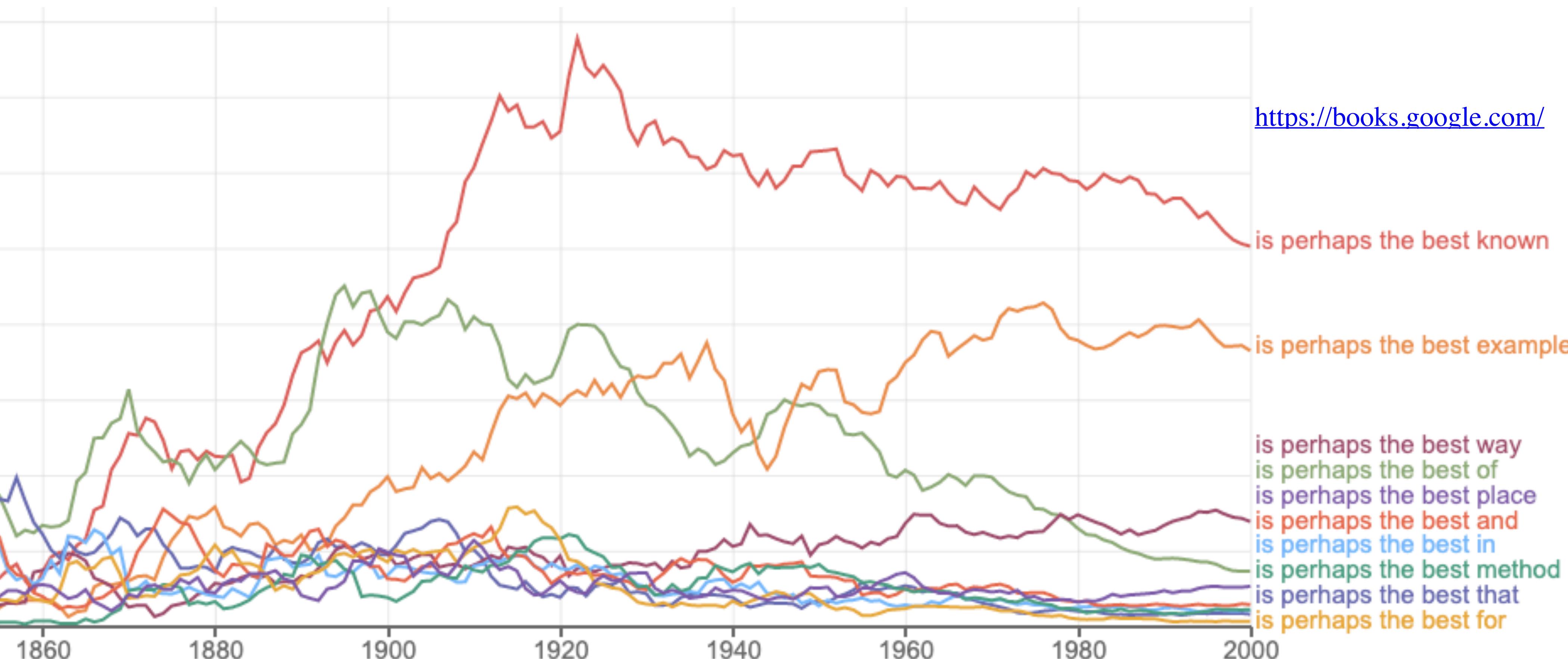
# You probably use a LM every day...

-  what is the mo
-  what is the mo - Google Search
-  what is the **most spoken language in the world**
-  what is the **most played game in the world**
-  what is the **most dangerous animal in the world**
-  what is the **most expensive car in the world**
-  what is the **moon made of**



# More examples

- ▶ .. is perhaps the best \_\_\_\_ ?



**So let's look deeper at LMs: from  
traditional LMs over neural LMs  
to contextualized embeddings**

# What is a Language Model (LM)?

- ▶ A computational model that can be used for either of the following two tasks is called a Language Model (LM):
  - ▶ to compute the probability of a text\*
  - $P(\text{today is a great day}) = ??$
  - ▶ to compute the probability of the next word
  - $P(\text{day} | \text{today is a great}) = ??$

\* (can be a text, sentence, phrase,...)

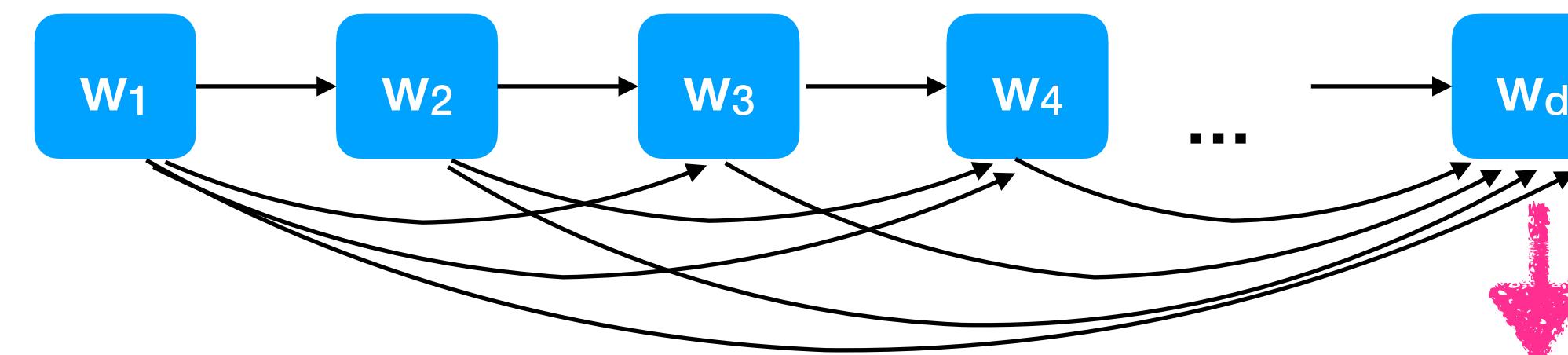
# A Language Model - Formally

- Given a sequence of words:  $(w_1, \dots, w_d)$
- An LM models the probability:  $P(w_1, \dots, w_d)$
- Without loss of generality (**Chain Rule**):

$$P(w_1, \dots, w_d) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2)\dots$$

$$= P(w_1) \prod_{i=2}^d P(w_i | w_1, \dots, w_{i-1})$$

conditioned on full history



$P(\text{Athens} | \text{an awesome summer school this year is in})$

# Markov assumption

- However, finding good estimates for the probabilities of longer sequences is a problem
- Therefore, a common assumption in a traditional LM is to make the Markov assumption:

$$P(x_1, \dots, x_d) = \prod_{i=1}^d P(x_i | x_1, \dots, x_{i-1}) \approx \prod_{i=1}^d P(x_i | \boxed{x_{i-(n-1)}}, \dots, x_{i-1})$$

$$\begin{aligned} p(\mathbf{w}) &= p(w_1) \times \\ &\quad p(w_2 | w_1) \times \\ &\quad p(w_3 | \cancel{w_1}, w_2) \times \\ &\quad p(w_4 | \cancel{w_1}, \cancel{w_2}, w_3) \times \\ &\quad \dots \end{aligned}$$

Markov: forget “distant” past

Valid for language? No...

Is it practical? Often.

**n-th order Markov assumption:**  
Only look at history of most recent n-1 words

-> n-grams



Adapted from Chris Dyer

# How to learn the parameters of an n-gram LM?

- **(Pre-deep learning LMs):** Learn a count-based **n-gram** (traditional) Language Model by **collecting statistics** of **n-grams** from a corpus to estimate the parameters of the model (maximum likelihood)
- **n-gram:** a chunk of consecutive words
  - Example: n=2 (bigram):  $P(x_i | x_{i-1})$

Collect statistics for “to buy”, “buy a”, “a house”.... with C=count():

$$P(w | \text{to}) = \frac{C(\text{to } w)}{C(\text{to})}$$

- Higher order n-grams, e.g. trigram LM:  $P(x_i | x_{i-2}, x_{i-1})$

# Unigram LM (1<sup>st</sup> order)

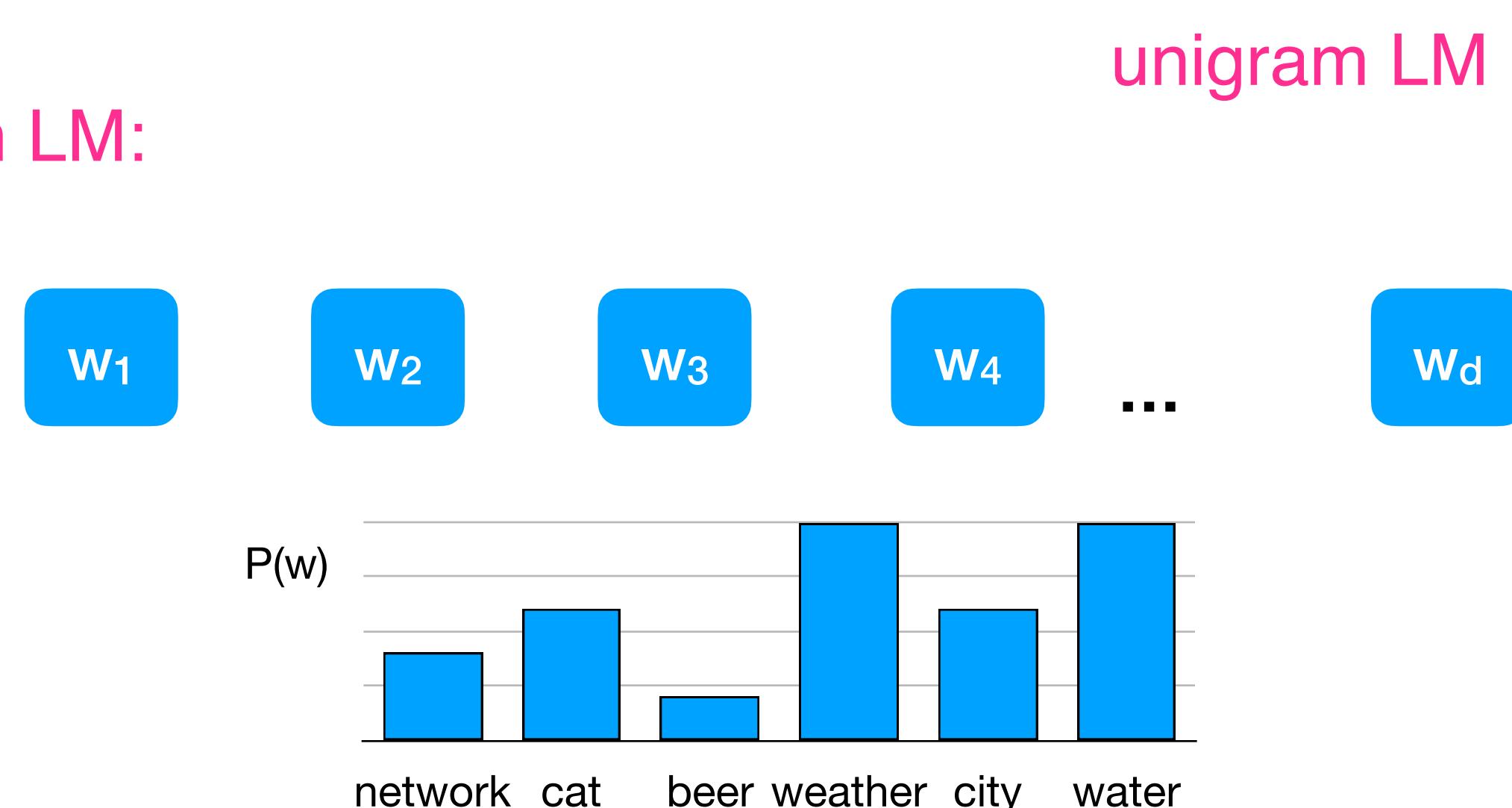
- n=1, 1st order Markov assumption, history (n-1): 0

$$P(w_1, \dots, w_d) = P(w_1)P(w_2)P(w_3)\dots$$

$$= \prod_{i=1}^d P(w_i)$$

Sampling from the unigram LM:

```
i = 0
repeat
  i ++
  xi ~ p(x)
until xi = STOP
return <x1, ..., xi>
```



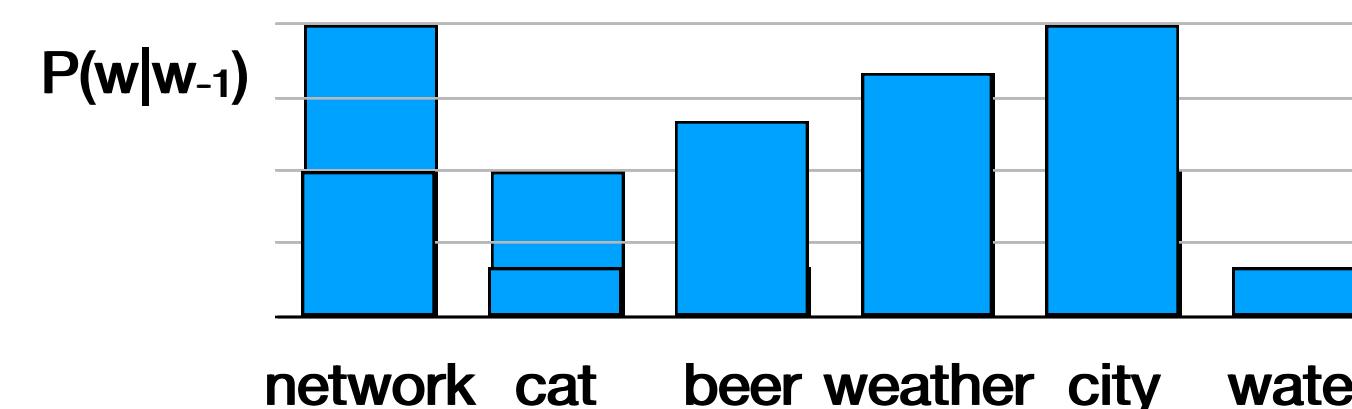
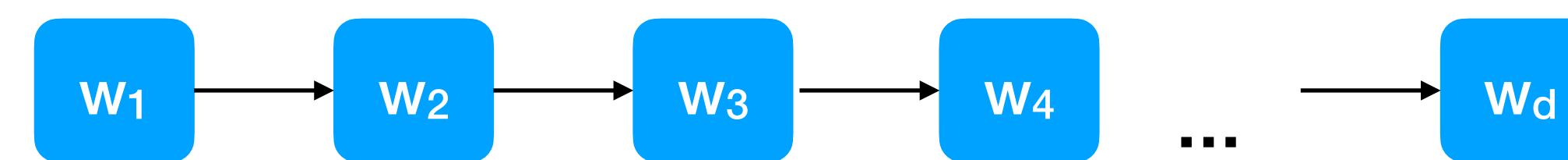
# Bigram Language Model

- $n=2$ , 2nd order Markov assumption, history ( $n-1$ ): 1

$$P(w_1, \dots, w_d) = P(w_1)P(w_2 | w_1)P(w_3 | w_2)\dots$$

$$= P(w_1) \prod_{i=2}^d P(w_i | w_{i-1})$$

$P(\text{center} | \text{city})$       bigram LM



# Sparsity problems with n-gram LMs

**Sparsity problem 1:** If “its water is so w” never occurred in the corpus: prob for w (and entire sequence) is 0!

What can we do? **Smoothing** (add small count to estimates)

$$P(w \mid \text{its water is so}) = \frac{C(\text{its water is so } w)}{C(\text{its water is so})}$$

**Sparsity problem 2:** If “its water is so” never occurred: prob for any w is 0.

What can we do? **Backoff** (condition on lower-level n-grams)

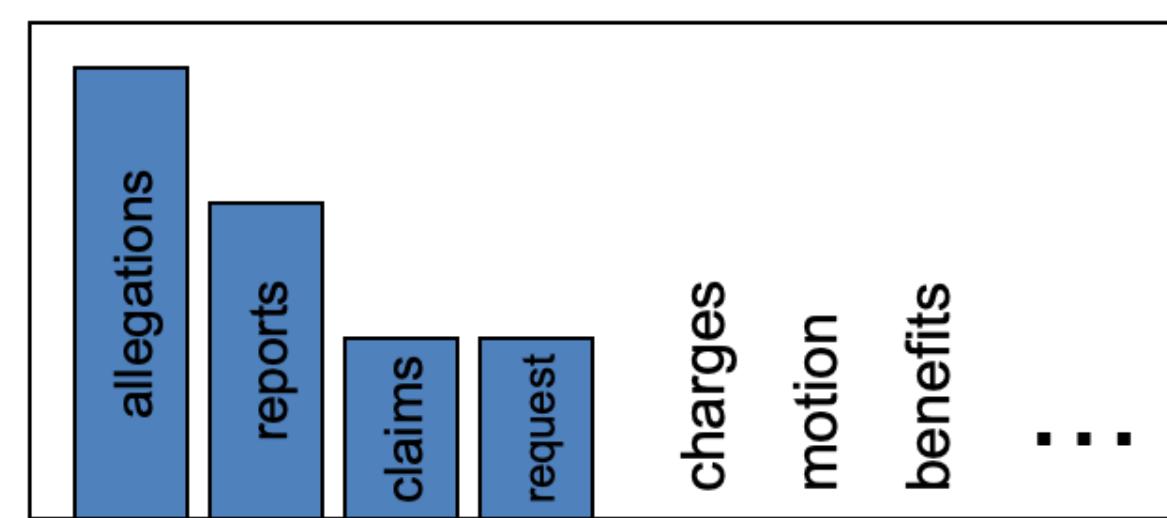
**C=count()**

**In general:** Increasing n-gram size makes sparsity problem worse.

# Smoothing

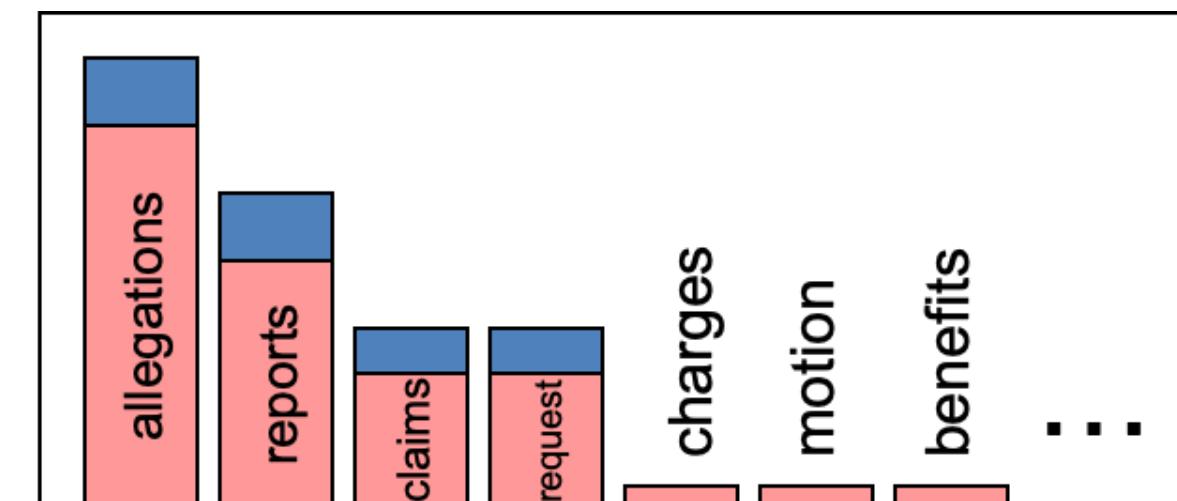
## Intuition

- Estimating statistics from sparse data
- Smoothing relocates probability mass to make generalization better



$P(w   \text{denied the})$
3 allegations
2 reports
1 claims
1 request
7 total

Smoothing



$P(w   \text{denied the})$
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total

# Smoothing

## Add-one Estimation

- Pretend we saw each word one more time than we did
- So, just add one to all counts
  - And don't forget to adjust normalization properly

$$p_{\text{MLE}}(x_i | x_{i-1}) = \frac{c(x_{i-1}, x_i)}{c(x_{i-1})} \longrightarrow p_{\text{Add-1}}(x_i | x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + |\mathcal{V}|}$$

- Also called Laplace smoothing

# Evaluating a LM

# Evaluation

## Testing

- How good is our LM?
- We must test the model on data it hasn't seen during learning
  - Otherwise — overfitting! 😱
- We need an evaluation metric — two options:
  - **Extrinsic:** focused on however the model will be used — for example, can it improve a transcription system?
  - **Intrinsic:** focused on the language model task — how good can the model assign probabilities to real unseen data?
- Ideally, the two correlate, but reality is more complex



# Intrinsic Evaluation

## The Shannon Game

- How well can we predict the next word?

When I eat pizza, I wipe of the \_\_\_\_\_

Many children are allergic to \_\_\_\_\_

I saw a \_\_\_\_\_

grease	0,5
sauce	0,4
dust	0,05
...	
mice	0,0001
...	
the	1E-100

- **Unigrams are terrible at this game (why?)**
- A better model of text is one which assigns a higher probability to the word that **actually** occurs

# Evaluation

## Perplexity

- The best language model is the one that is best at predicting the test set → will give test sentences the highest probability
- Perplexity is the inverse probability of the test set, normalized by the number of words:
  - Given a set of test sentences  $D'$  with a total of  $m$  words:

$$PP(D') = p(D')^{-1/m} = \left( \prod_{\bar{x} \in D'} p(\bar{x}) \right)^{-1/m}$$

- In practice, we work in log space:

$$PP(D') = 2^{-\frac{1}{m} \sum_{\bar{x} \in D'} \log_2 p(\bar{x})}$$

- Lower perplexity is better

# Evaluation

## Perplexity of a Uniform Model

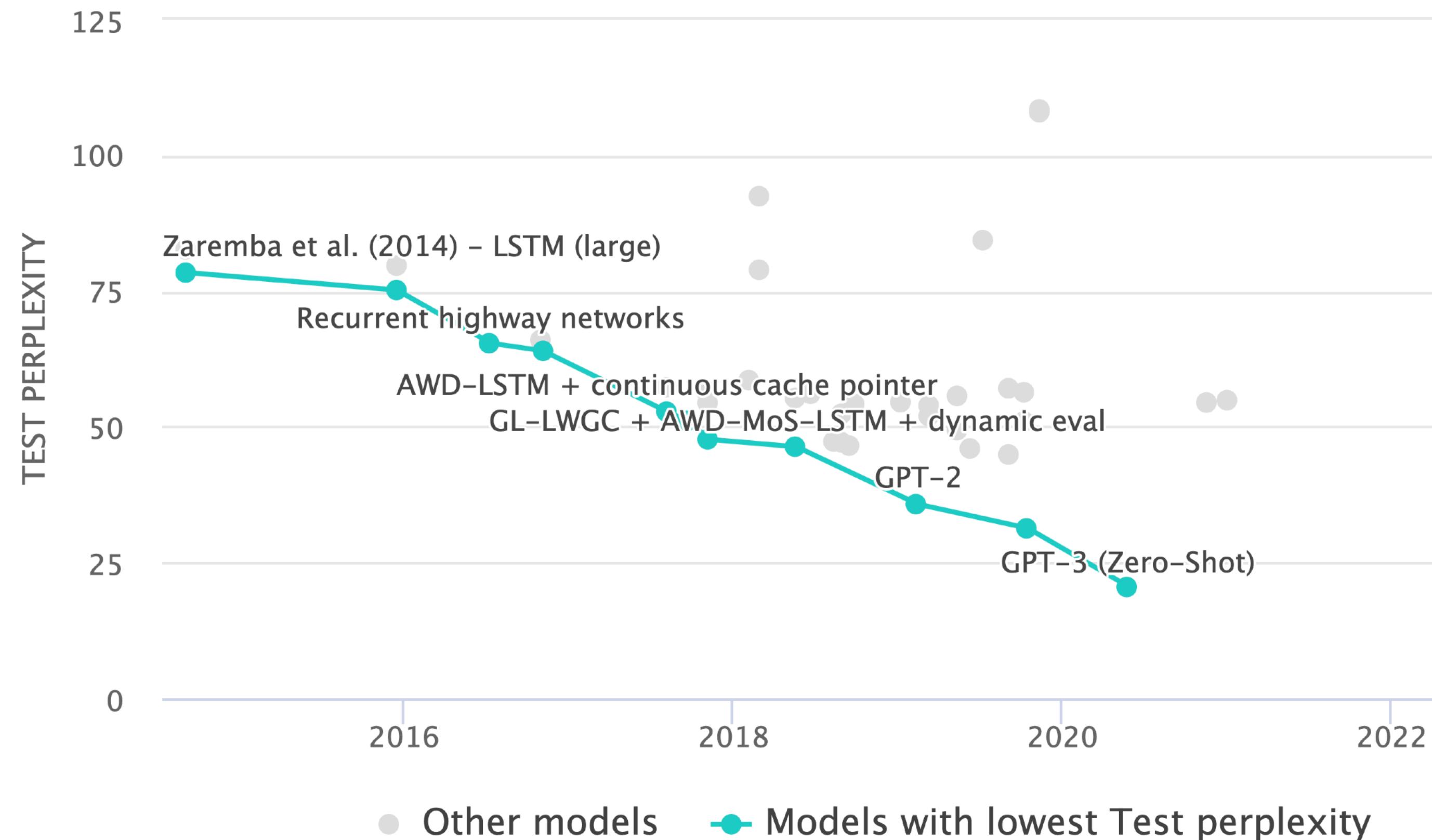
- Under a uniform distribution perplexity will be the vocabulary size
- Assume  $M$  sentences consisting of  $m$  random digits,  $|\mathcal{V}| = 10$
- What is the perplexity of this data for a model that assigns  $p(\cdot) = \frac{1}{10}$  to each digit

$$\begin{aligned} PP &= 2^{-\frac{1}{m} \sum_{i=1}^M \log_2(\frac{1}{10})^{|\bar{x}^{(i)}|}} \\ &= 2^{-\frac{1}{m} \sum_{i=1}^M |\bar{x}^{(i)}| \log_2 \frac{1}{10}} \\ &= 2^{-\log_2 \frac{1}{10}} = 2^{-\log_2 10^{-1}} = 10 \end{aligned}$$

- Perplexity is weighted equivalent **branching factor**

# Evaluation

## Perplexities of Contemporary Models



# Further issues with n-gram LMs

- ▶ What about similar words?

- ▶ *she bought* a bicycle
  - ▶ *she purchased* a bicycle



cannot share strength  
among similar words

- ▶ Long-distance dependencies?

- ▶ *for programming* she yesterday purchased her own brand new *laptop*
  - ▶ *for running* she yesterday purchased her brand new *sportswatch*



cannot handle long-distance  
dependencies

# **Neural LMs**

# Neural Language Models

- LMs so far: count-based estimates of probabilities
  - Counts are brittle and generalize poorly, so we added smoothing
- The quantity that we are focused on estimating (e.g., for tri-gram model):

$$p(\bar{x}) = \prod_{i=1}^n p(x_i | x_{i-1}, x_{i-2})$$

- Can we use neural networks for this task? What would it give us?

# Neural Language Models

## A Very Simple Approach

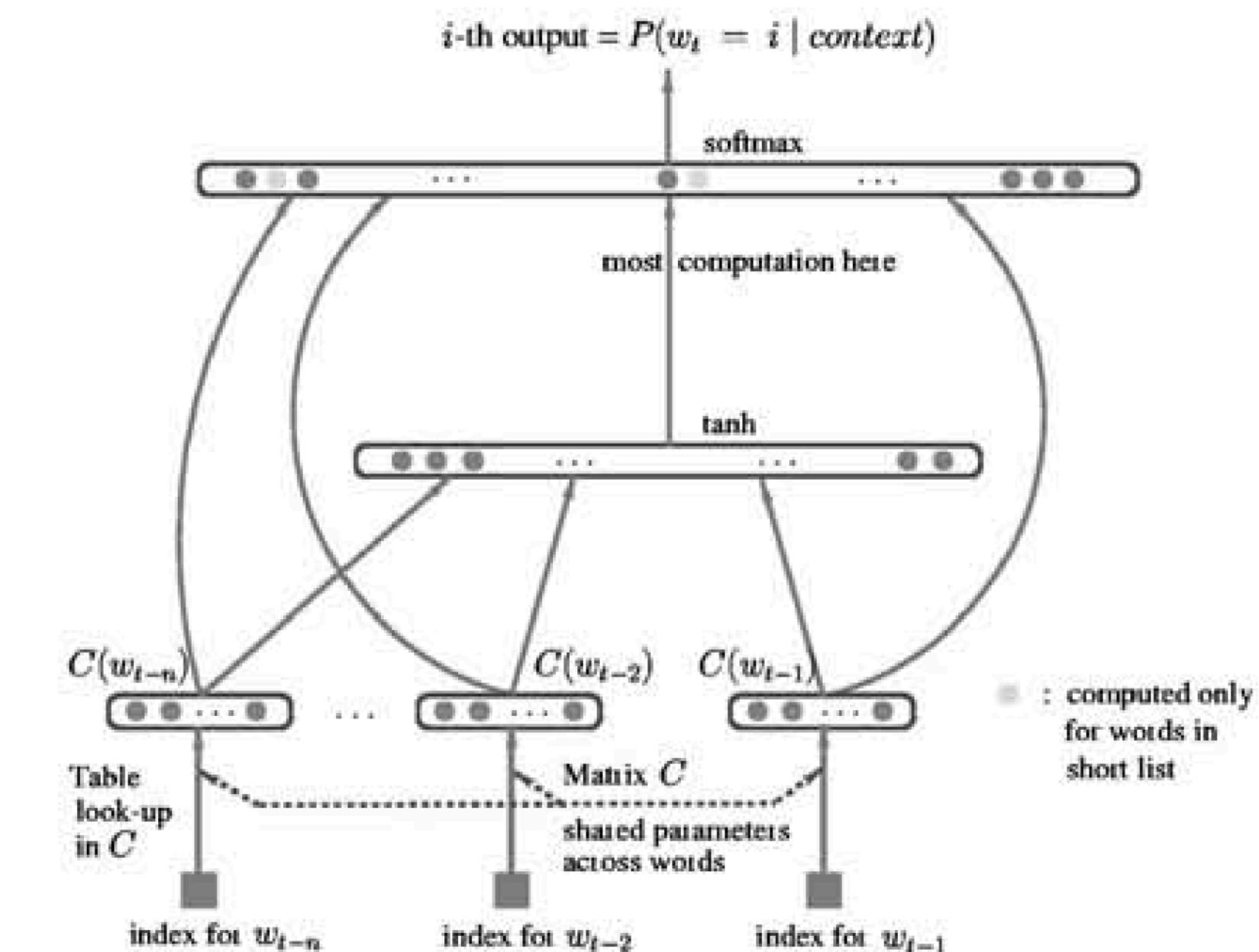
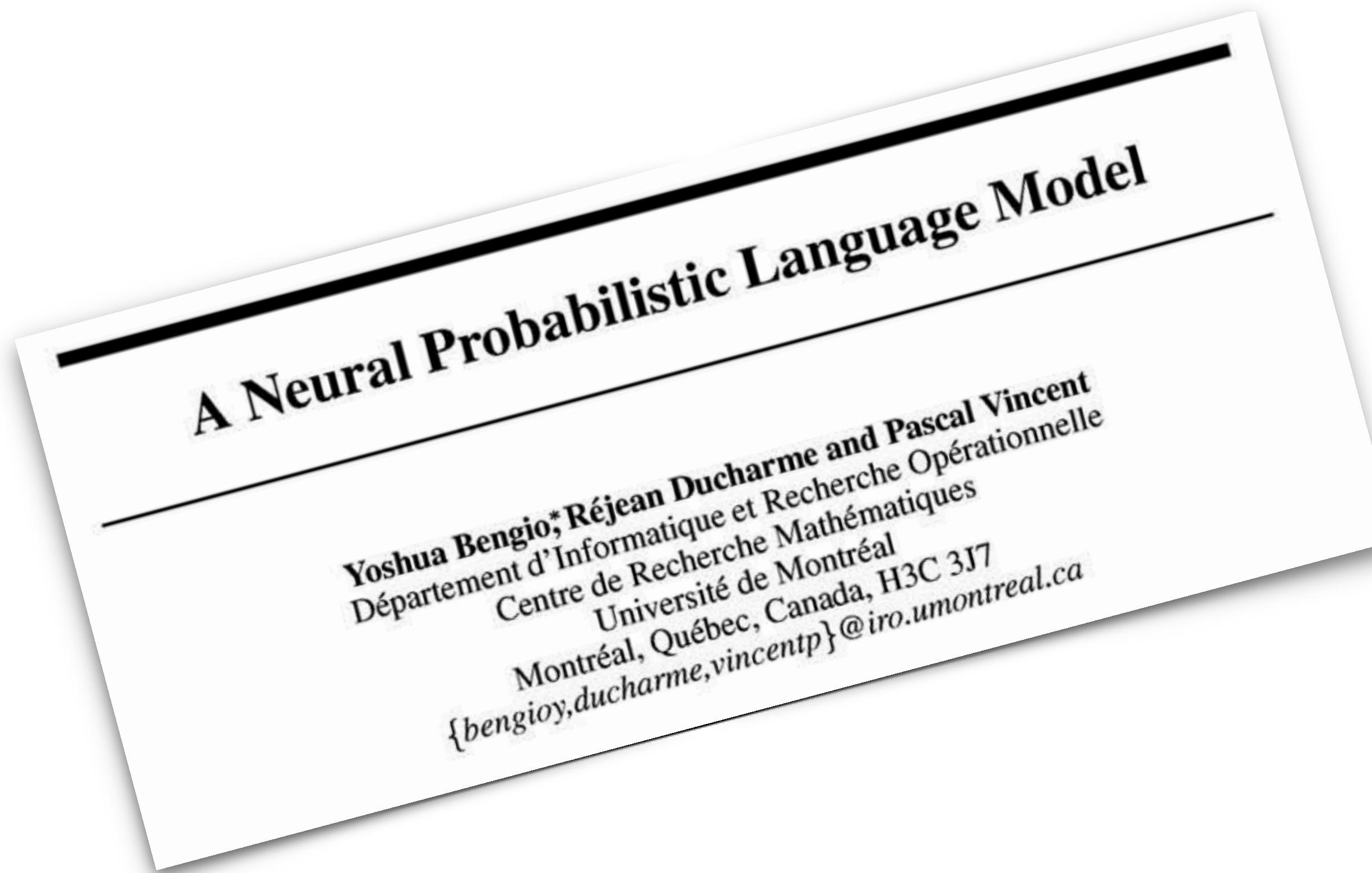
- Instead of having count-based distributions, we parameterize them

$$p(x_i | x_{i-1}, x_{i-2}; \theta)$$

- How would we model this with a neural network?
  - Hint: so far, only learned about MLPs

# Early Neural LMs (1/2)

- An early solution: a **neural probabilistic language model** (Bengio et al., 2000)



feed-forward neural network, n-grams with distributed representations

# Neural Language Models

## A Very Simple Approach

- A simple FFNN-ish model

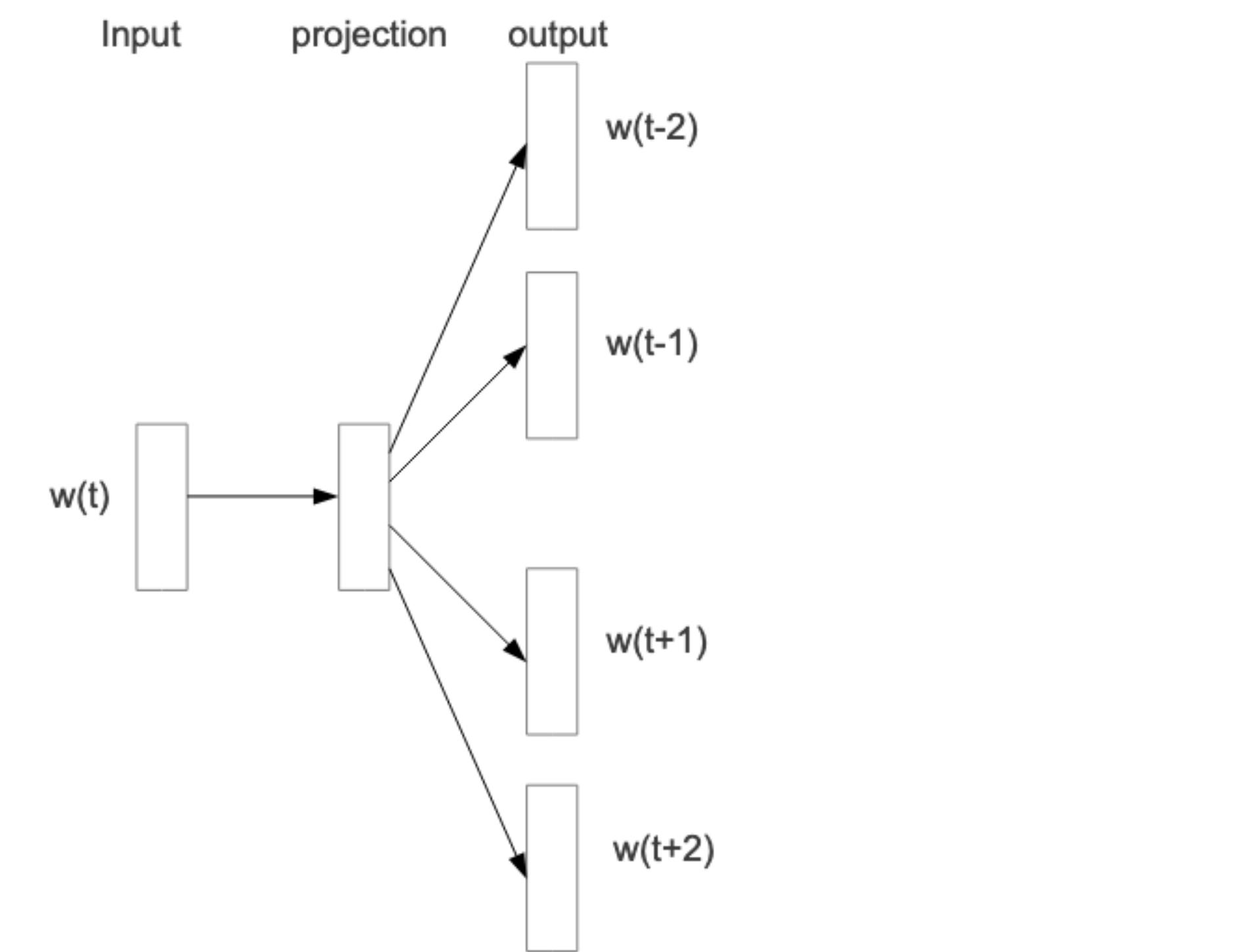
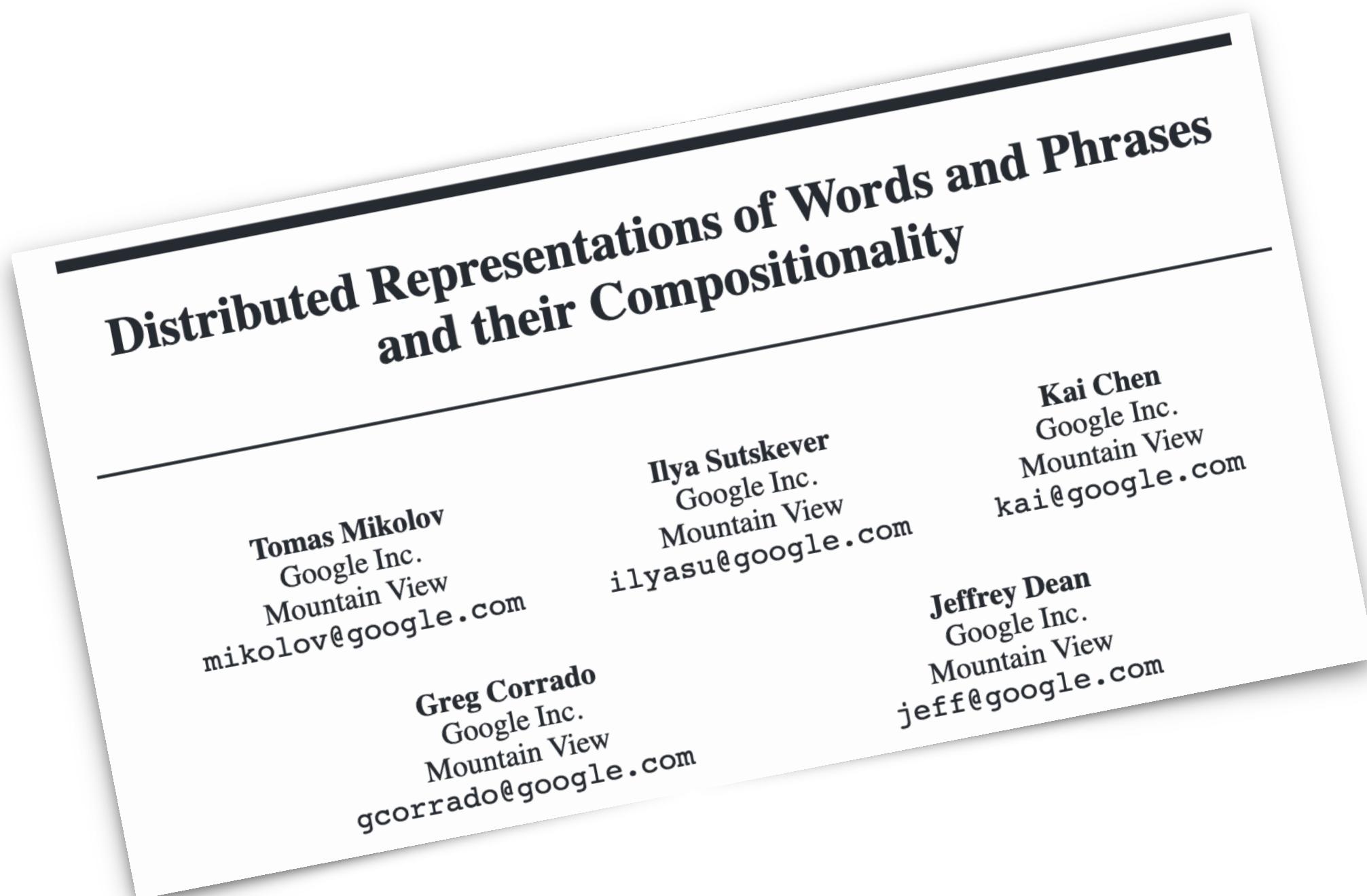
$$\begin{aligned} p(x_i = w | x_{i-1}, x_{i-2}; \theta) &= \text{softmax}(\mathbf{y})_w \\ \mathbf{y} &= \mathbf{U} \tanh(\mathbf{W}\mathbf{e} + \mathbf{b}_1) + \mathbf{b}_2 \\ \mathbf{e} &= [\phi(x_{i-1}); \phi(x_{i-2})] \end{aligned}$$

where  $\phi$  is an embedding function, and  $\theta = (\mathbf{b}, \mathbf{d}, \mathbf{W}, \mathbf{U}, \mathbf{H}, \mathbf{C}, \phi)$

- The parameters  $\theta$  are estimated by maximizing the log probability of the data
- During inference, you compute the neural network every time you need a value from the probability distribution

# Early Neural LMs (2/2)

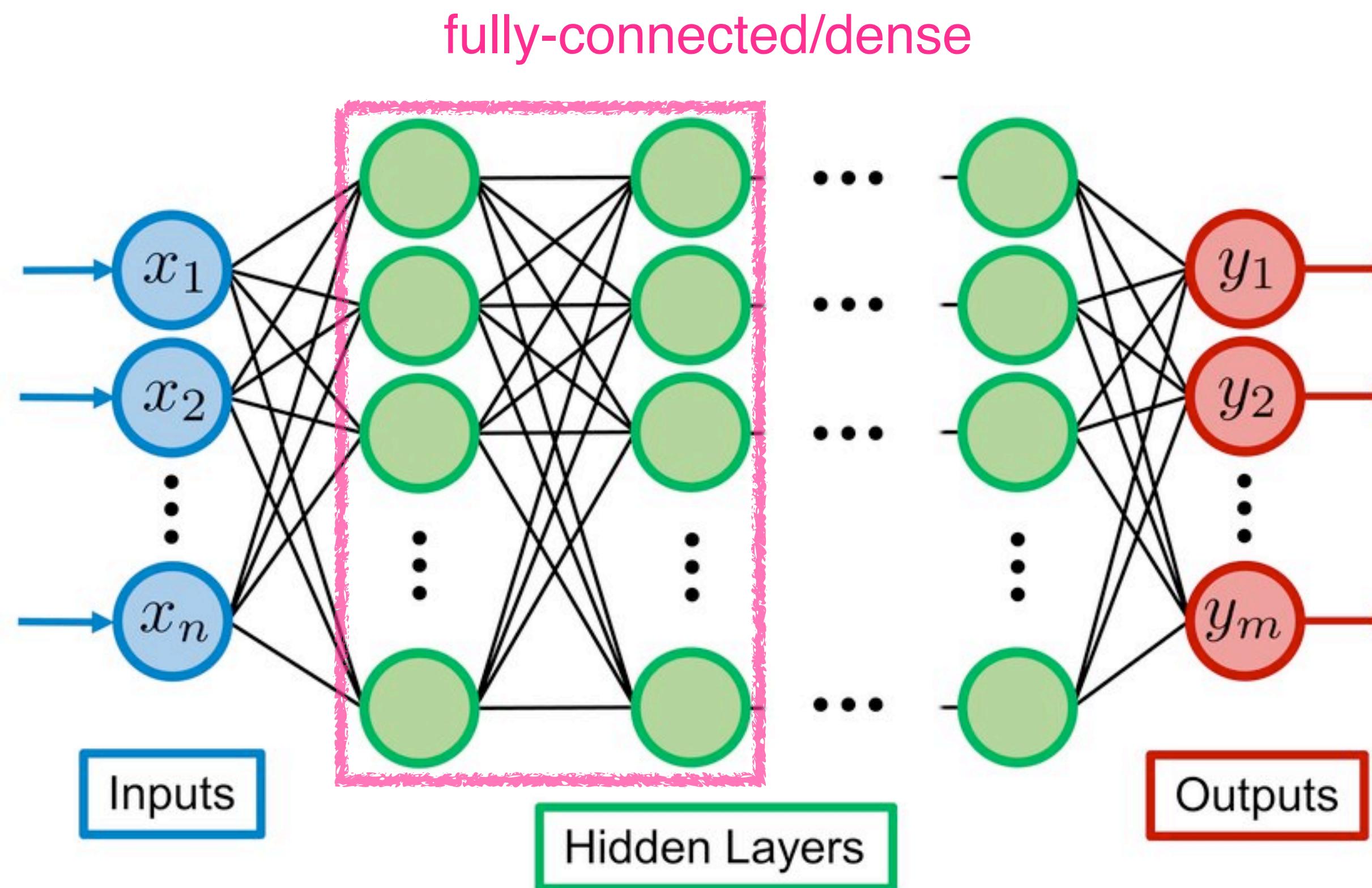
- Another early solution: **word2vec** (Mikolov et al., 2013) models the probability of a word given a context [See Ryan's lecture 1 yesterday for details]; is a family of methods. Learn a word embedding directly. Example: Skip-gram:



feed-forward neural network, skip-grams

# **Recap: Feed-forward Neural Networks**

# Recap: A Feed-Forward Neural Network (FFNN) - Graphical View



- A (dense / fully-connected) feed-forward neural network
  - AKA a Multi-layer Perceptron (MLP)

# Hidden Layer Connections - Notation

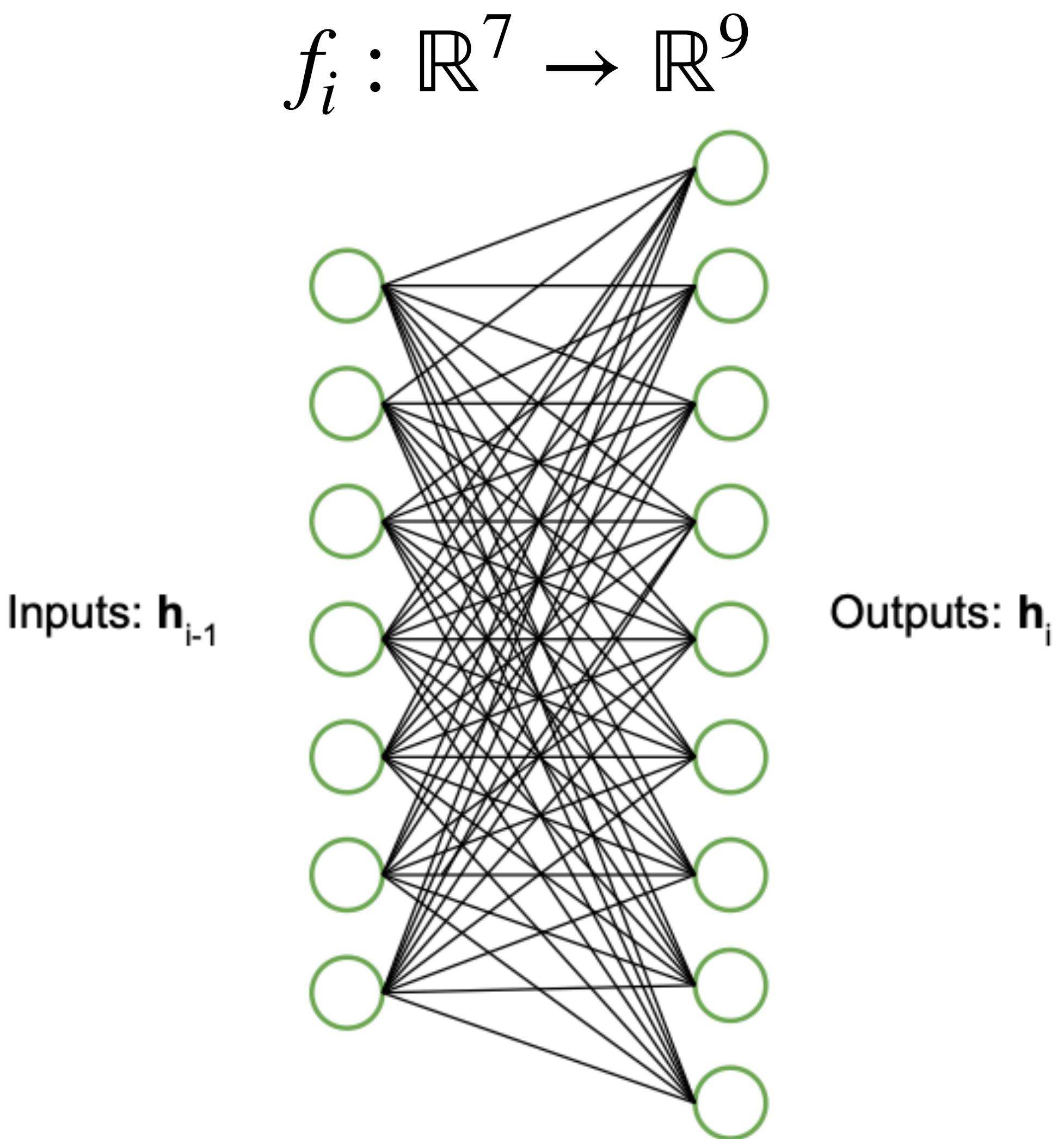
$\mathbf{h}_i \in \mathbb{R}^{D_i}$  Hidden layer with  $D_i$  dimensions, e.g.  $\mathbb{R}^9$

$$\mathbf{h}_i = \sigma_i(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i)$$

$\mathbf{W}_i \in \mathbb{R}^{D_i \times D_{i-1}}$

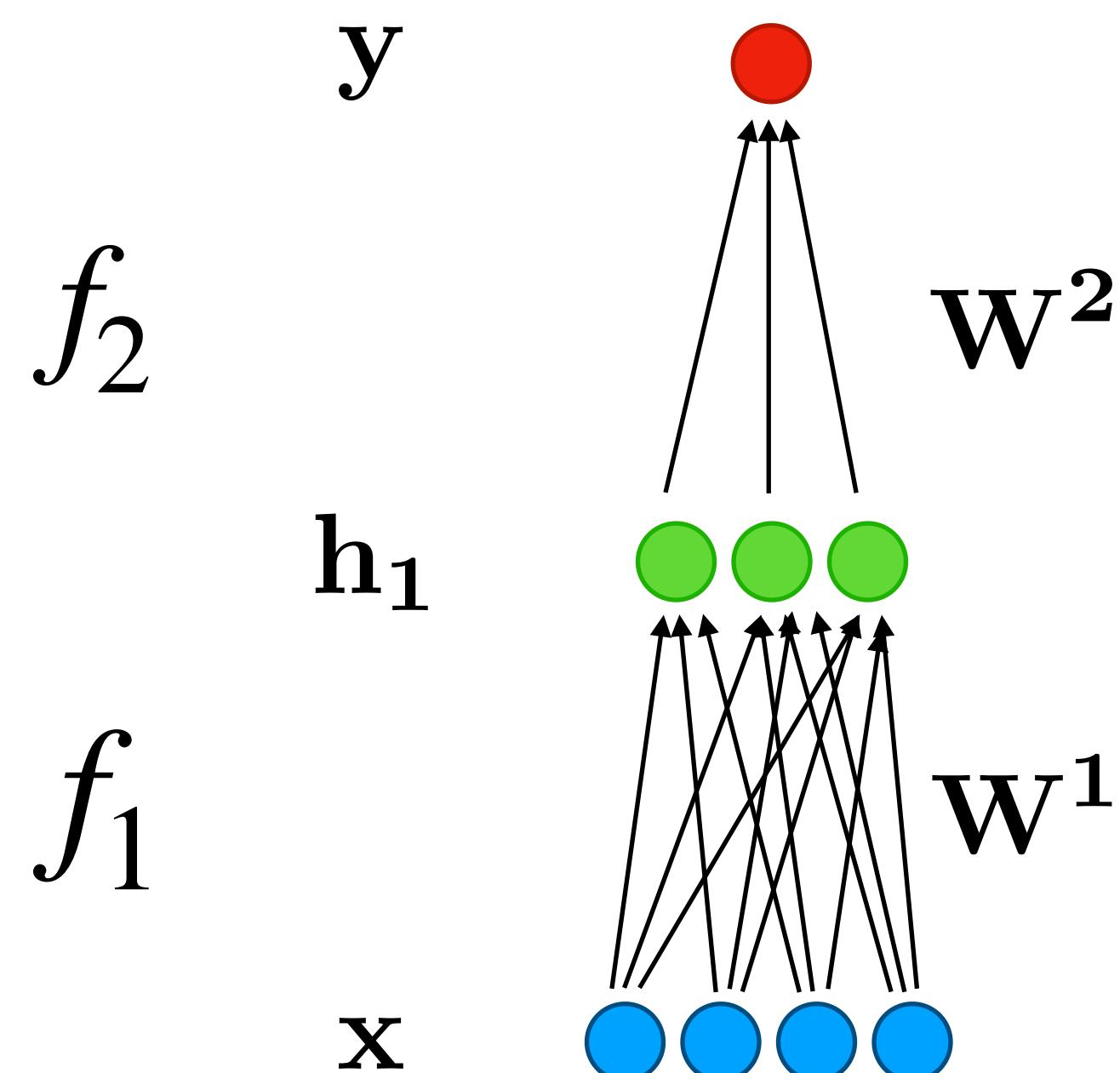
$\mathbf{b}_i \in \mathbb{R}^{D_i}$

$\sigma_i$  Layer's non-linear activation function



# Feed-Forward Neural Network - Functional Application View

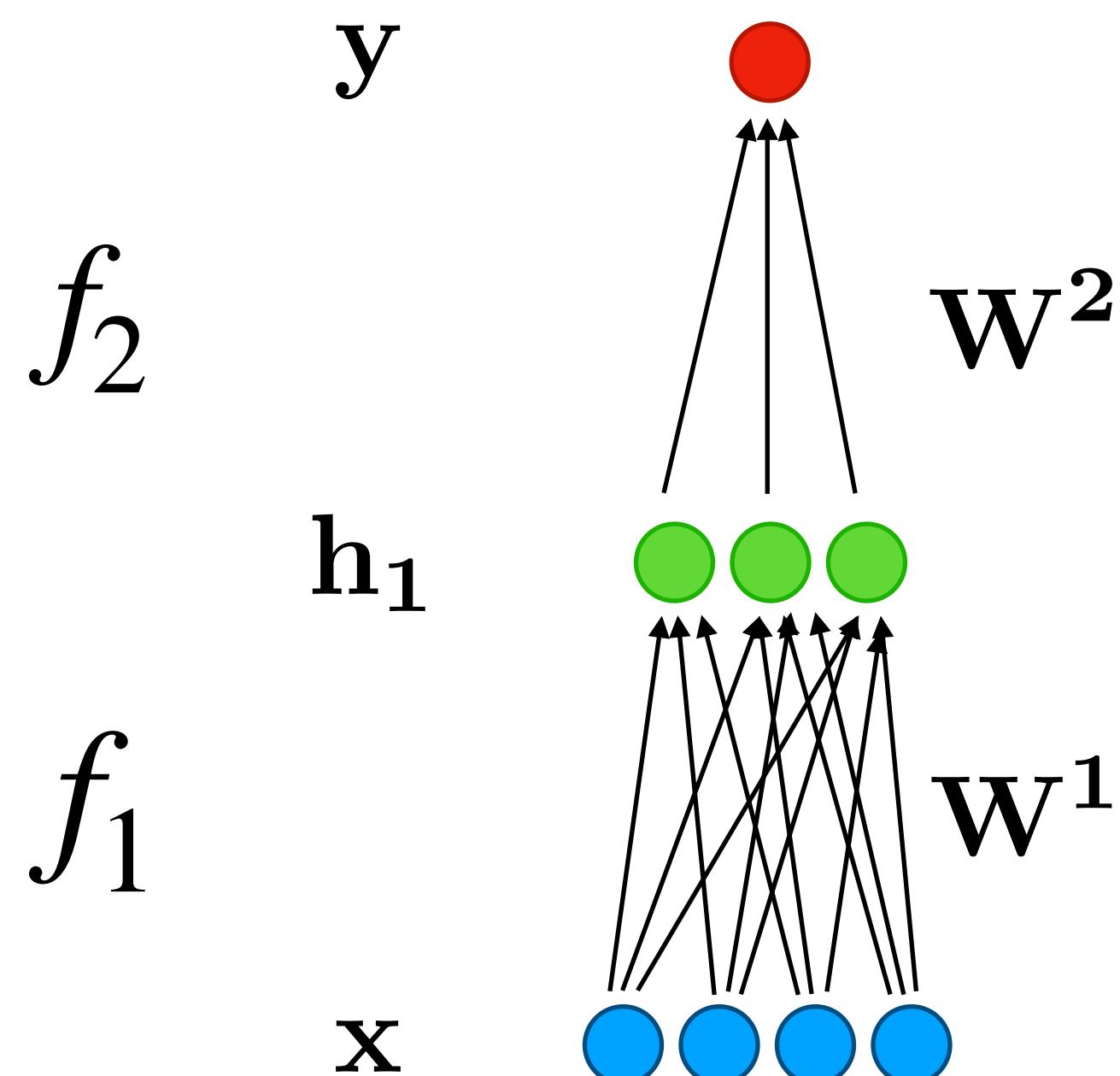
$$\hat{y} = f_2(f_1(x))$$



# Feed-Forward Neural Network - Algebraic View

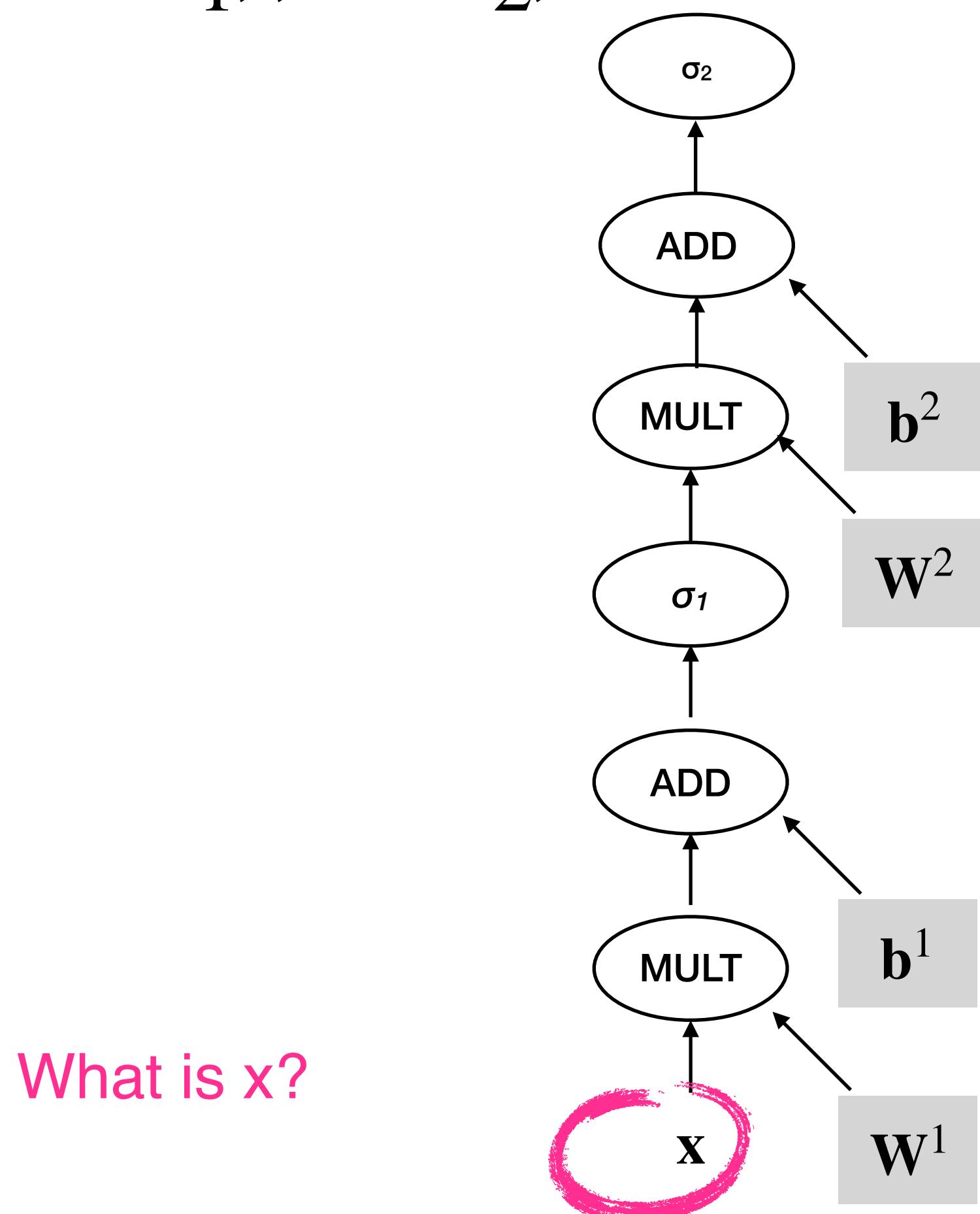
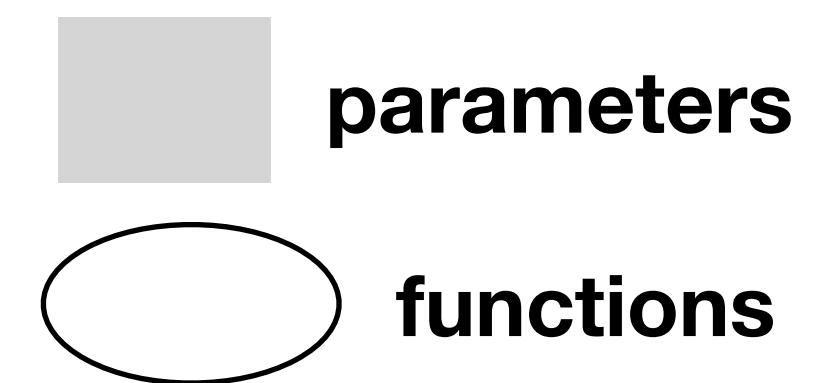
$$\hat{y} = f_2(f_1(x))$$

$$= \sigma_2(\mathbf{W}_2(\sigma_1(\mathbf{W}_1\mathbf{x} + b_1)) + b_2)$$



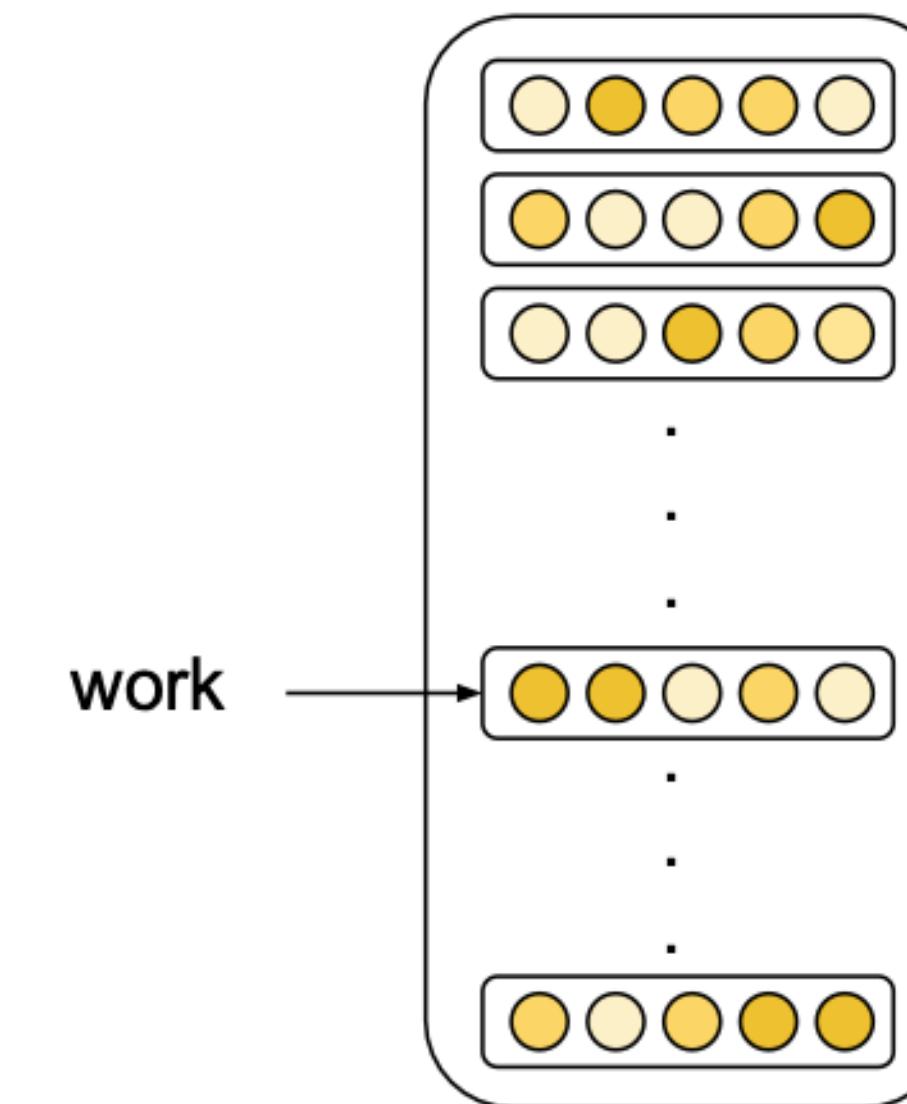
# Feed-Forward Neural Network - Computational Graph View

$$\sigma_2(W_2(\sigma_1(W_1x + b_1)) + b_2)$$



What is  $x$ ?

# Recap: Input layer as Embedding Layer



- Input is a word  $x \in \mathbb{R}^D$  for all  $x \in \mathcal{V}$
- We store these in a  $|\mathcal{V}| \times D$  look up table
  - These are the model *word embeddings*
  - AKA embedding layer; word look-up table; ...

# Recap: One-hot encoding

- Sparse high-dimensional vector of dimension  $|V|$  (=size of vocabulary)

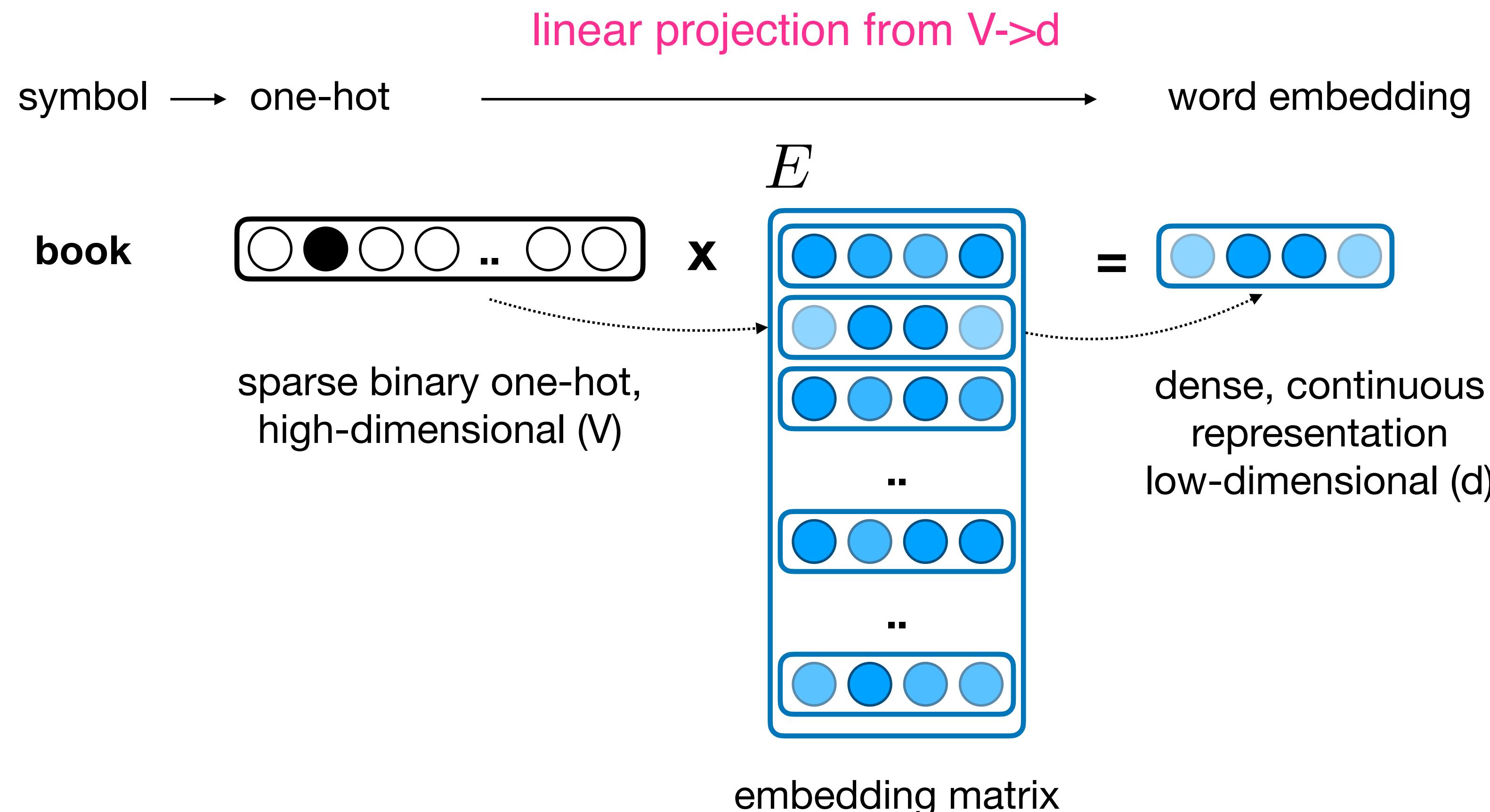
**Symbol** (word, char,..)

**one-hot vector**  
(length  $V$ , one entry is 1)

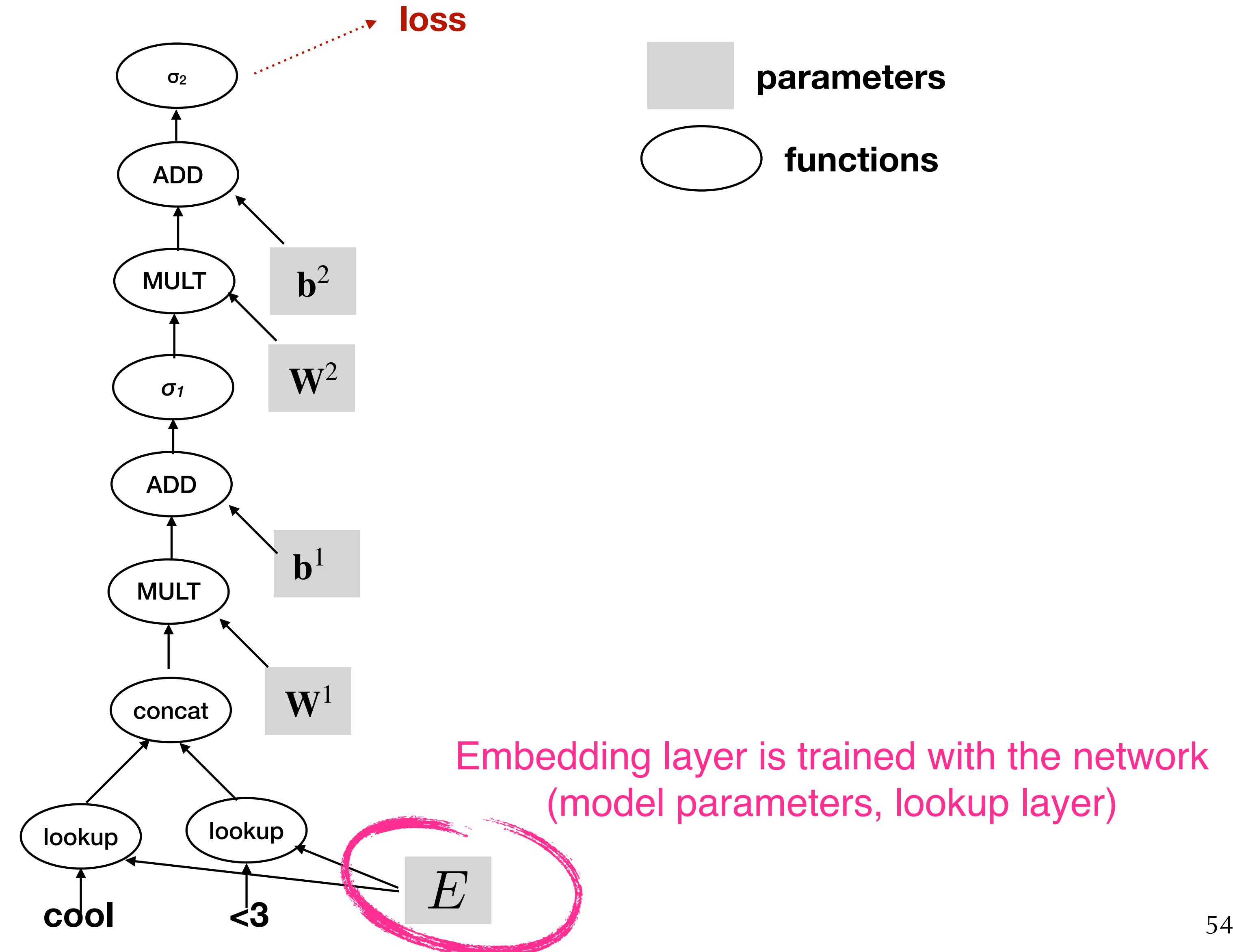
**yellow**



# Lookup: Representing a symbol



# Computational Graph View



# Putting it all together: A window-based neural LM

output distribution

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

hidden layer

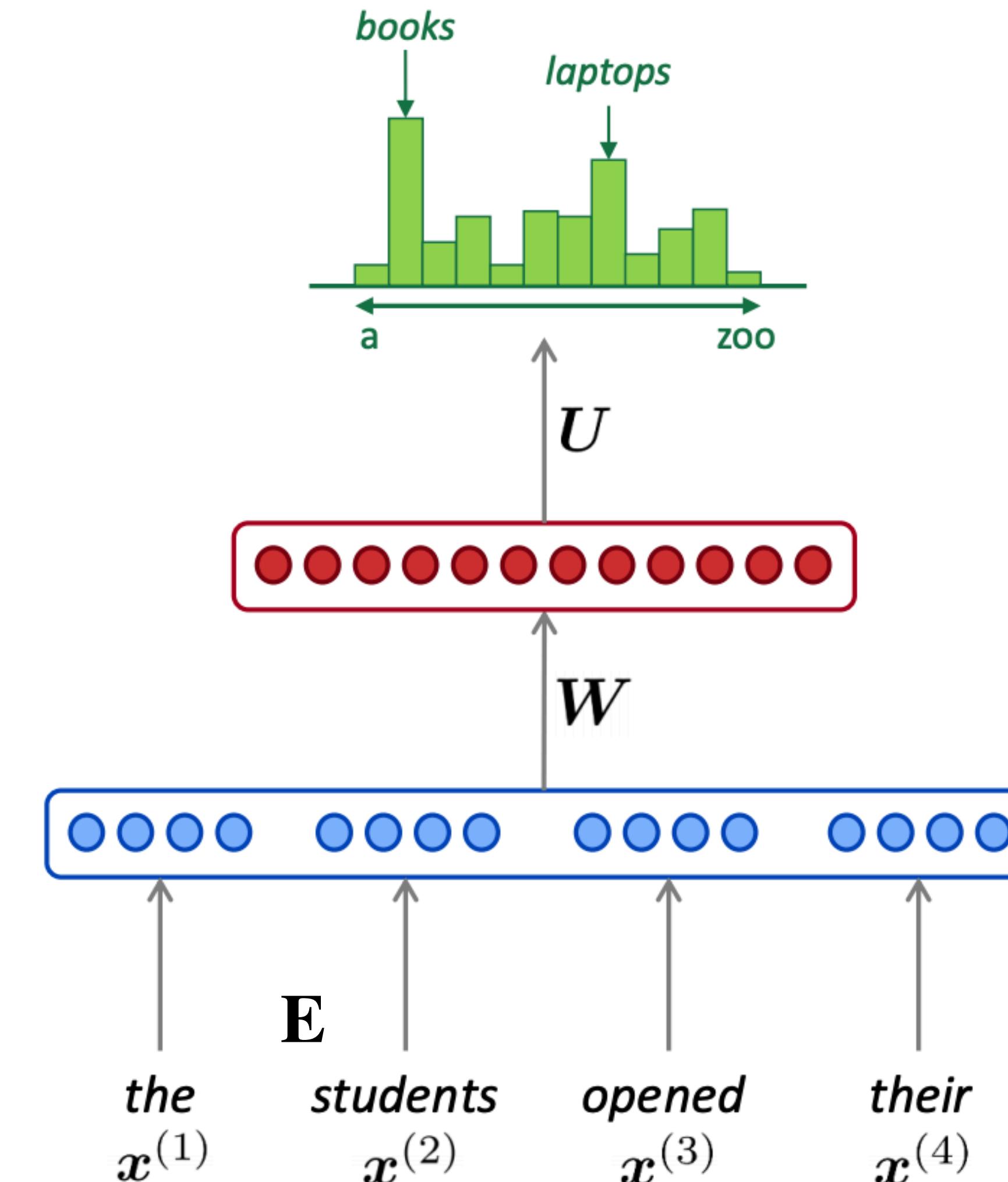
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

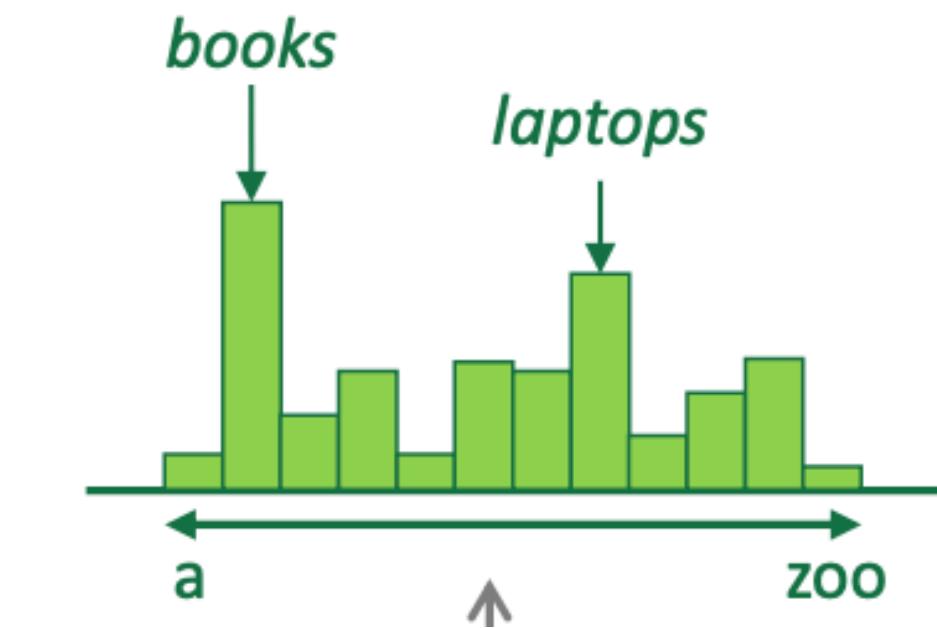
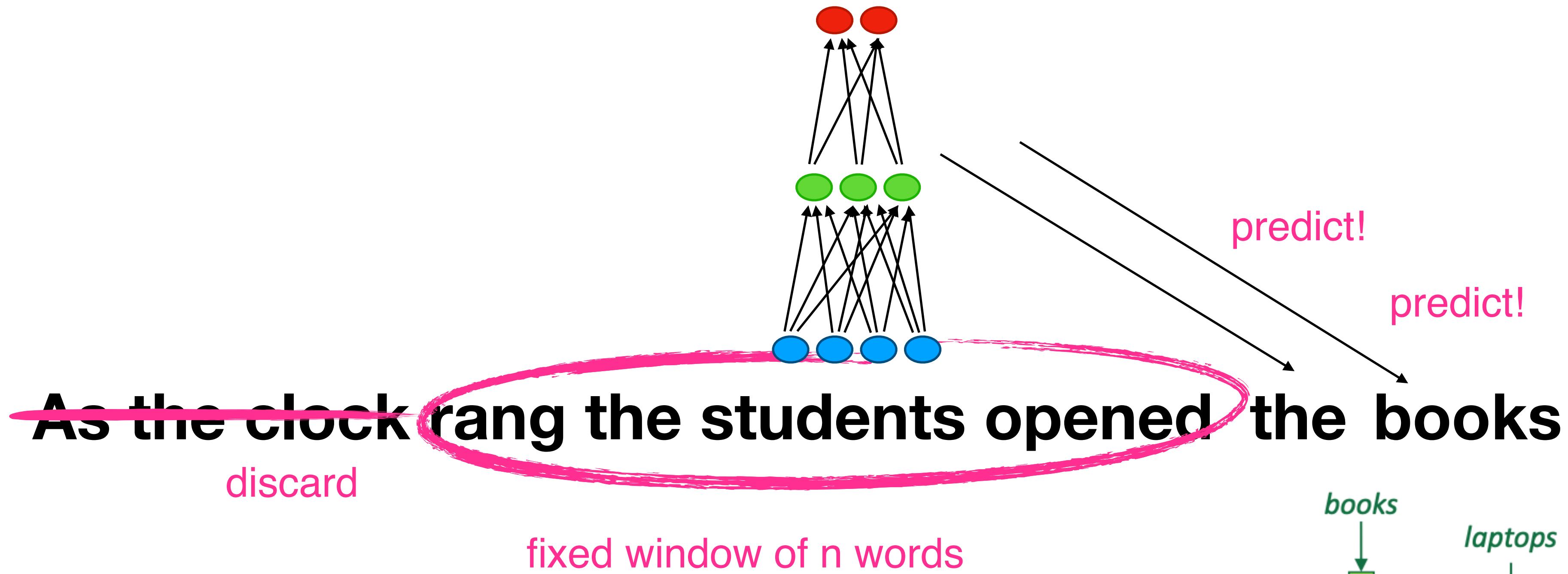
$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$



# Window-based neural LM via FFNN

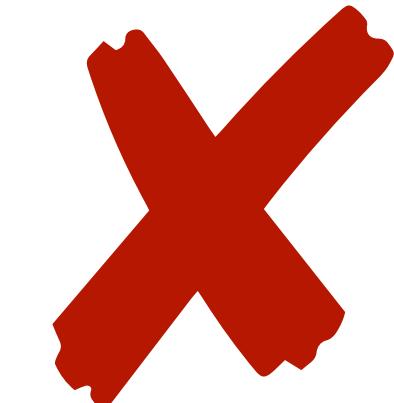


# Training the Neural FFNN-based LM

- Iteratively move the n-gram window through a very large corpus to predict the next word at each time step
  - Cross-entropy loss (negative log-likelihood):
 
$$L = -\log p(w_t | w_{t-1} \dots w_{t-n+1})$$
  - Note: typically very large vocabulary (softmax)
    - Workaround: negative sampling (lecture 1)
- output distribution  
 $\hat{y} = \text{softmax}(U\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$
- hidden layer  
 $\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$
- concatenated word embeddings  
 $\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$
- words / one-hot vectors  
 $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$
-

# Does a neural FFNN-LM solve these issues?

- ▶ Can it handle similar words?
  - ▶ *she bought* a bicycle
  - ▶ *she purchased* a bicycle
- ▶ Long-distance dependencies?
  - ▶ *for programming she yesterday purchased her own brand new laptop*
  - ▶ *for running she yesterday purchased her brand new sportswatch*





# Neural Language Models

## An MLP with No Markov Assumption

- Without the Markov assumption, the model is

$$p(\bar{x}) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

- We need to model the parameterized distribution

$$p(x_{i+1} | x_1, \dots, x_i; \theta)$$

- How can we do this with the tools we already know?

# Neural Language Models

## An MLP with No Markov Assumption

- We need to model the parameterized distribution

$$p(x_{i+1} | x_1, \dots, x_i; \theta)$$

- We can just treat the context as a bag of words
  - Then it doesn't matter how long it is
  - A very simple example (two layer MLP)

$$\mathbf{h} = \tanh(\mathbf{W}' \frac{1}{i} \sum_{j=1}^i \phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} | x_1, \dots, x_i) = \text{softmax}(\mathbf{W}'' \mathbf{h} + \mathbf{b}'')$$

# Neural Language Models

## An MLP with No Markov Assumption

- Why is this a terrible idea?
  - Order matters a lot in language 
- ▶ “it was not **good**, it was actually quiet **bad**”
- ▶ “It was not **bad**, it was actually quiet **good**”
- But it worked so well for text categorization ... 
- What may work for tasks that just require focusing on salient words (e.g., topic categorization), is not sufficient for language models (i.e., next-word prediction)

# FFNN: Any problems?

PROBLEM:  
we need fixed input size!

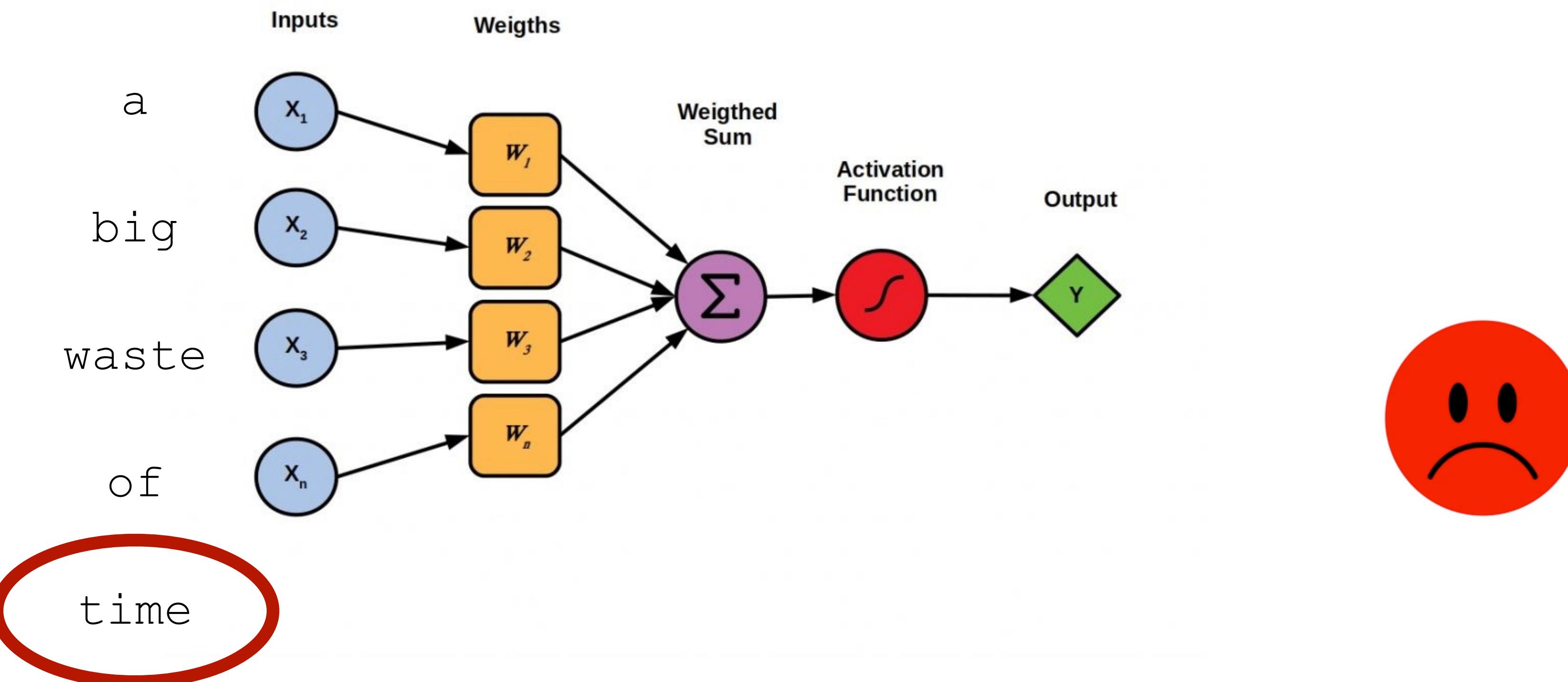


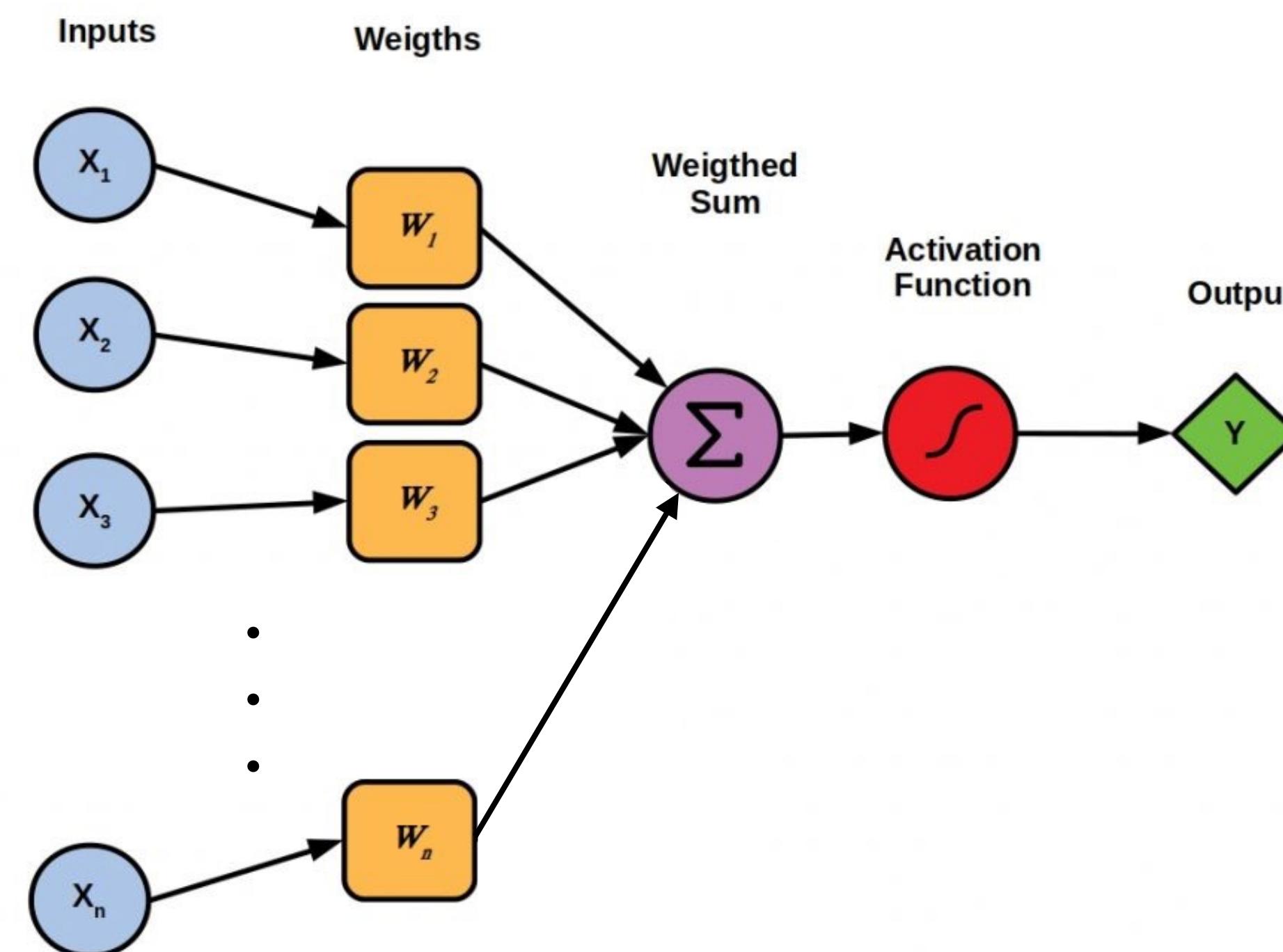
Illustration by Elisa Bassignana.

# FFNN: What is still missing?

PROBLEM:  
it does not consider the order well

it  
was  
not  
good  
,

it  
was  
actually  
quite  
bad



# Variable Length Inputs

- Feed-forward neural networks assume fixed-length inputs, but texts are not fixed lengths

the cat sits there

the sleepy cat sits there

the sleepy cat which chased the dog sits there

# Dealing with Variable Length Inputs

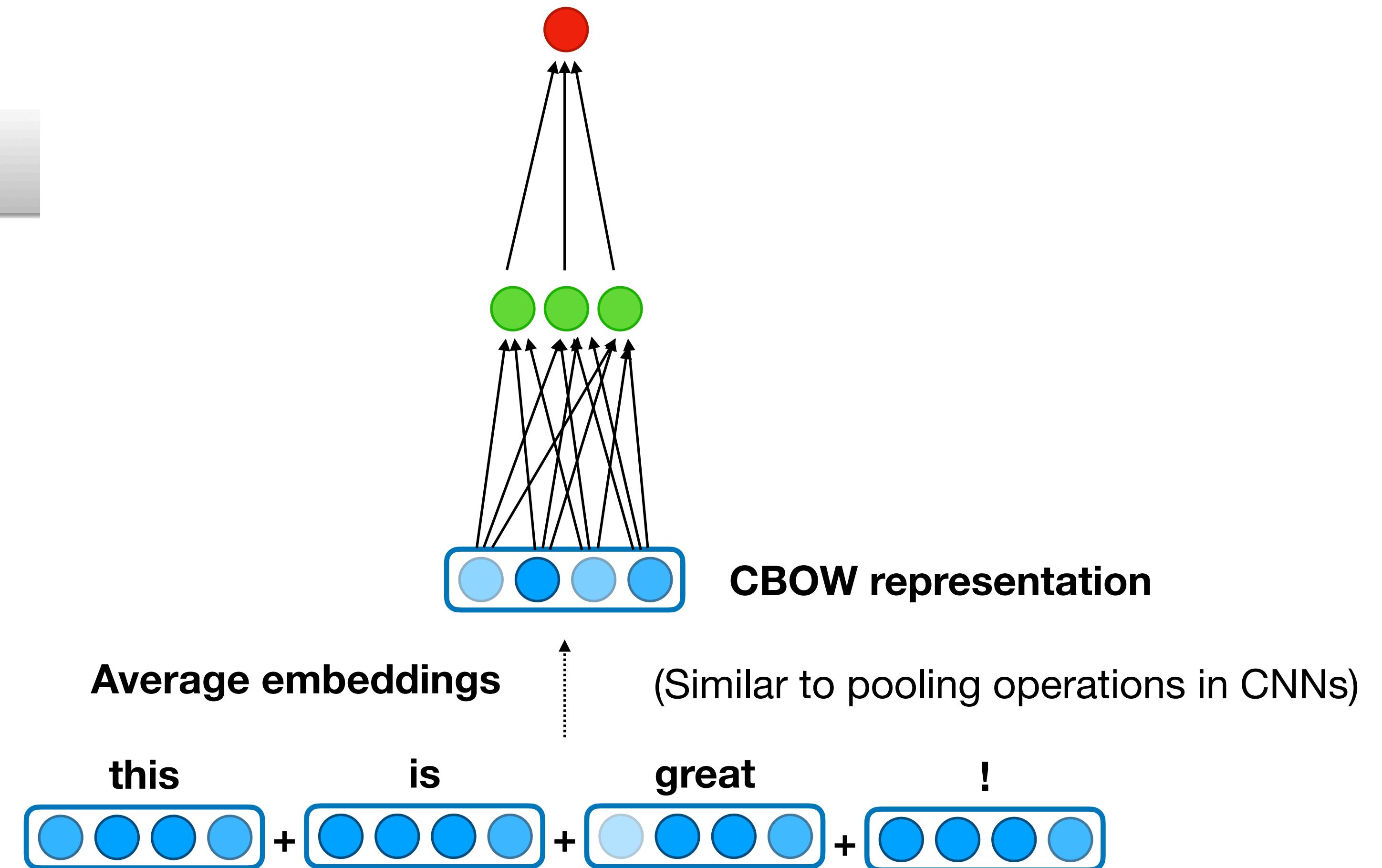
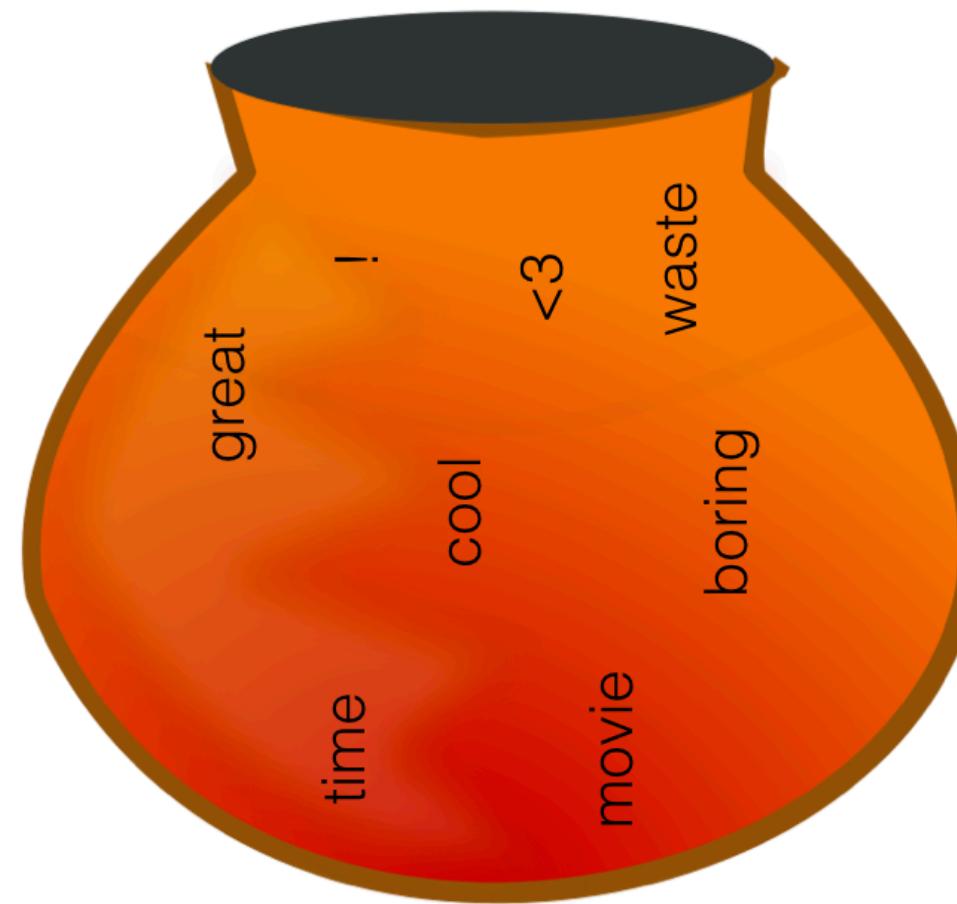
- **Options** (1-3 covered in lecture 1):
  - (1) Truncate text length at fixed K
  - (2) Average embeddings (pooling)

# Average embedding: Continuous Bag-of-words (CBOW)

Example input document:

this is great !

bag-of-words

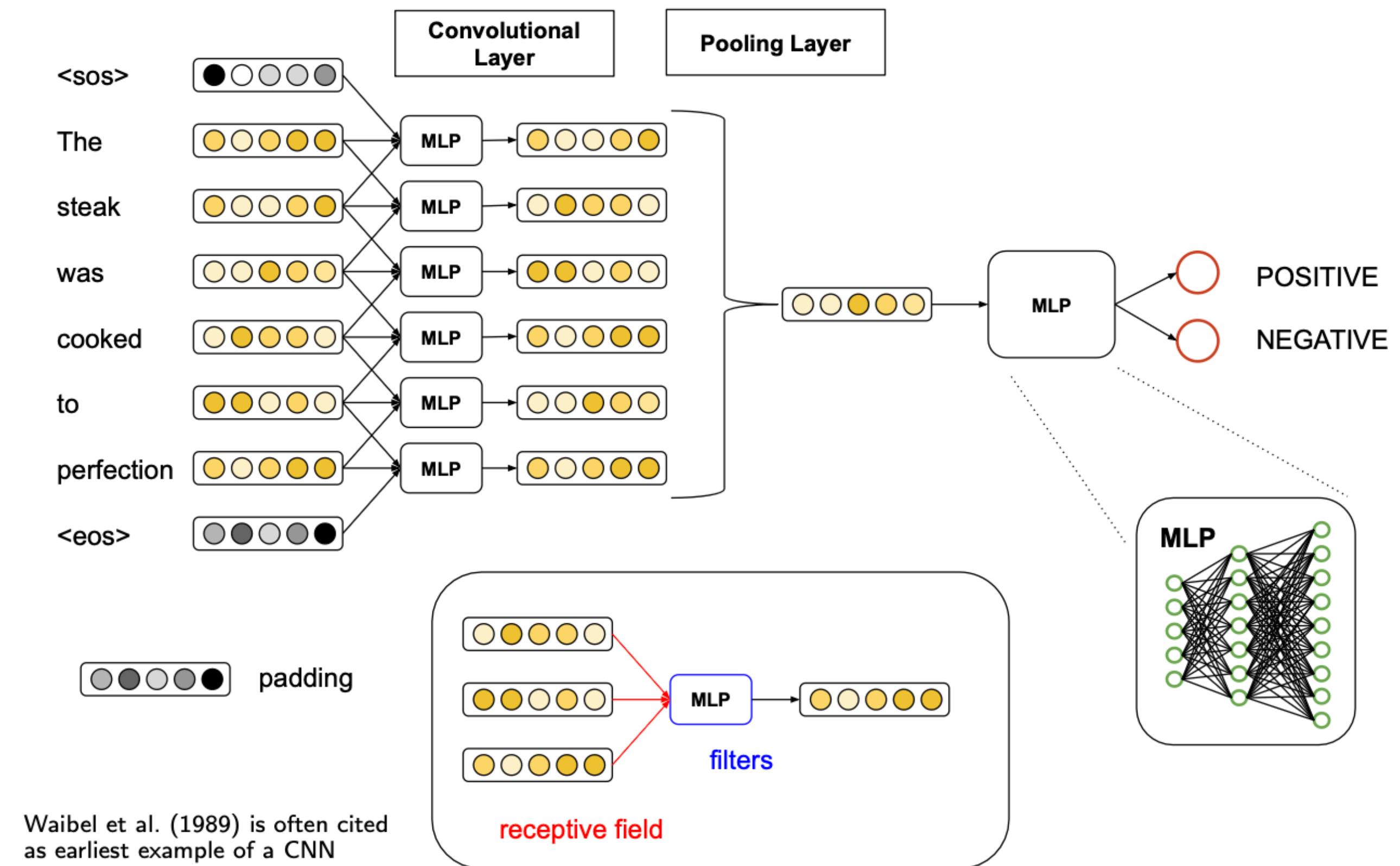


# Dealing with Variable Length Inputs

- More Options:

(3) **Convolutional neural network (CNN)**  
illustration from last lecture by Ryan:

(4) **Recurrent neural network (RNN)**  
covered next

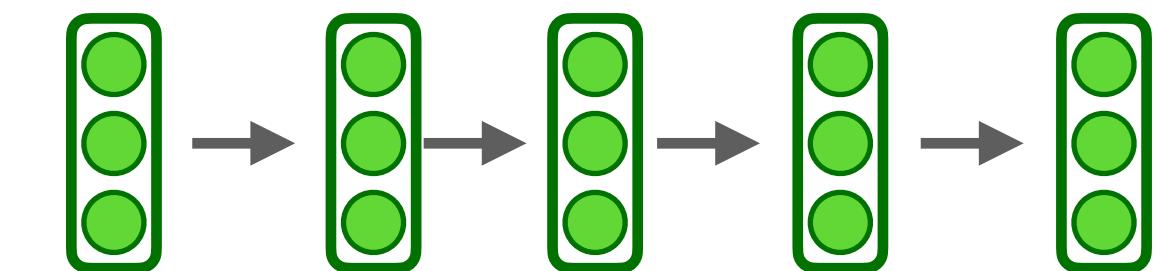


# Representations & Beyond FFNNs

Part II

# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution

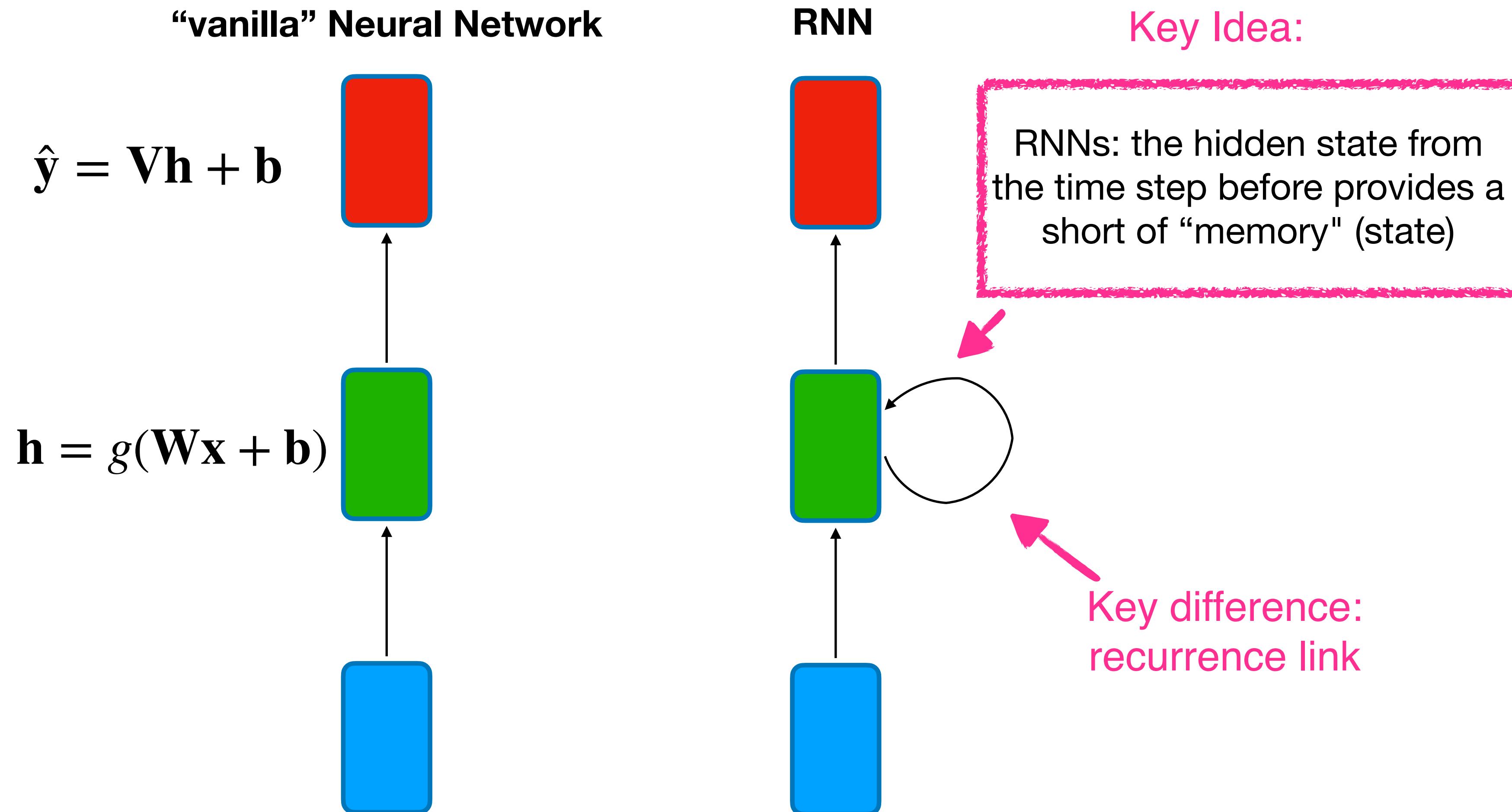


# RNNs

# Recurrent Neural Networks (RNNs)

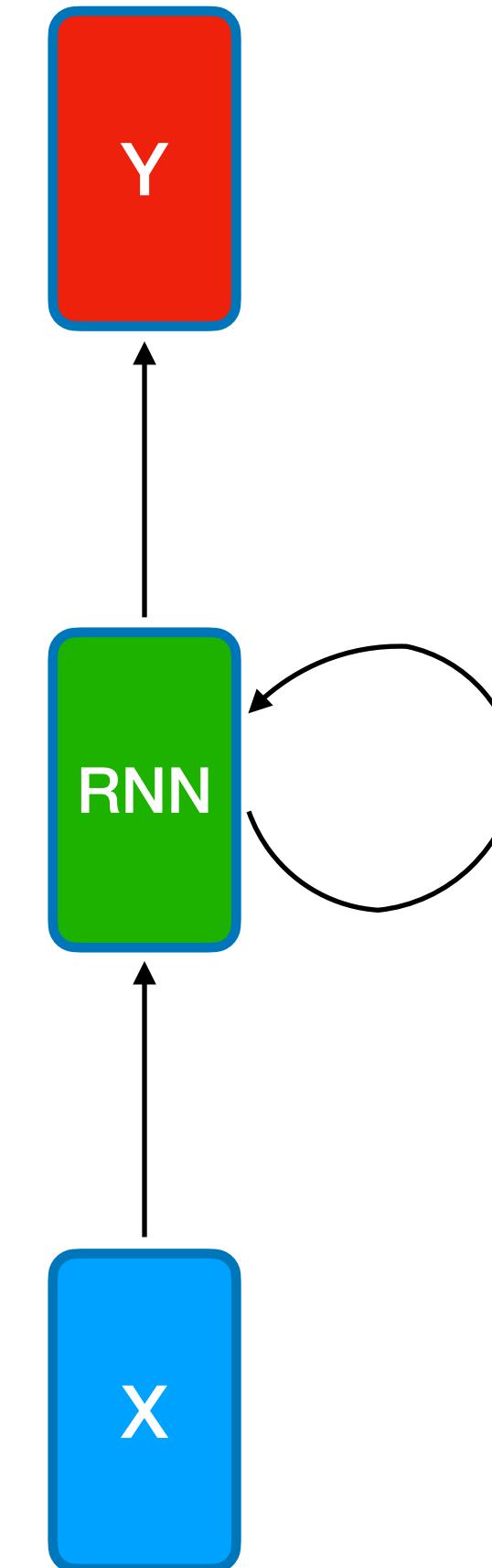
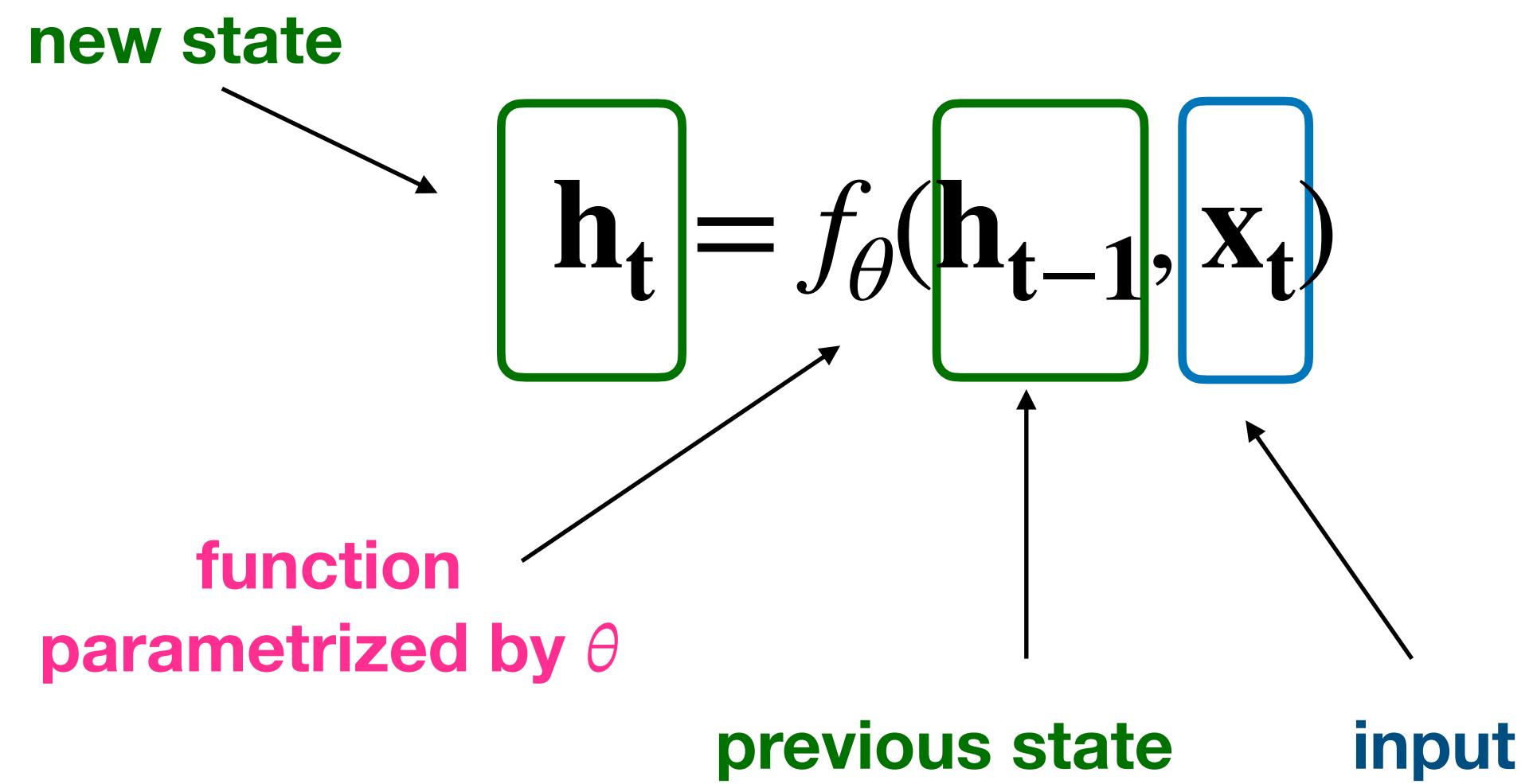
- ▶ RNNs are a family of neural networks
- ▶ Very good at modelling sequential input of **variable length**
  - ▶ Unlike CBOW, they **model the order** in the sequence
  - ▶ Unlike vanilla CNNs, they **can deal with long-distance dependencies** (especially the gated RNN variants)
  - ▶ RNNs as LMs do **not need to make the Markov** assumption

# Recurrent Neural Networks (RNNs)



# A closer look: inside an RNN layer

- We process a sequence  $x$  by applying a recurrence formula **at every time step  $t$**
- **Critically**, this does not impose a fixed-length input!

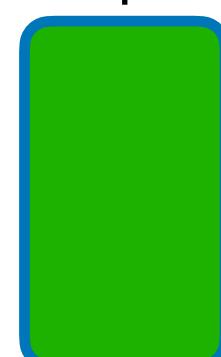


# Recurrent Neural Networks (RNNs)

RNN

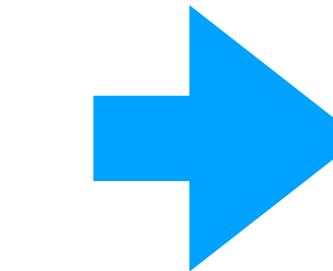


$$\hat{y}_t = Vh_t + b_2$$

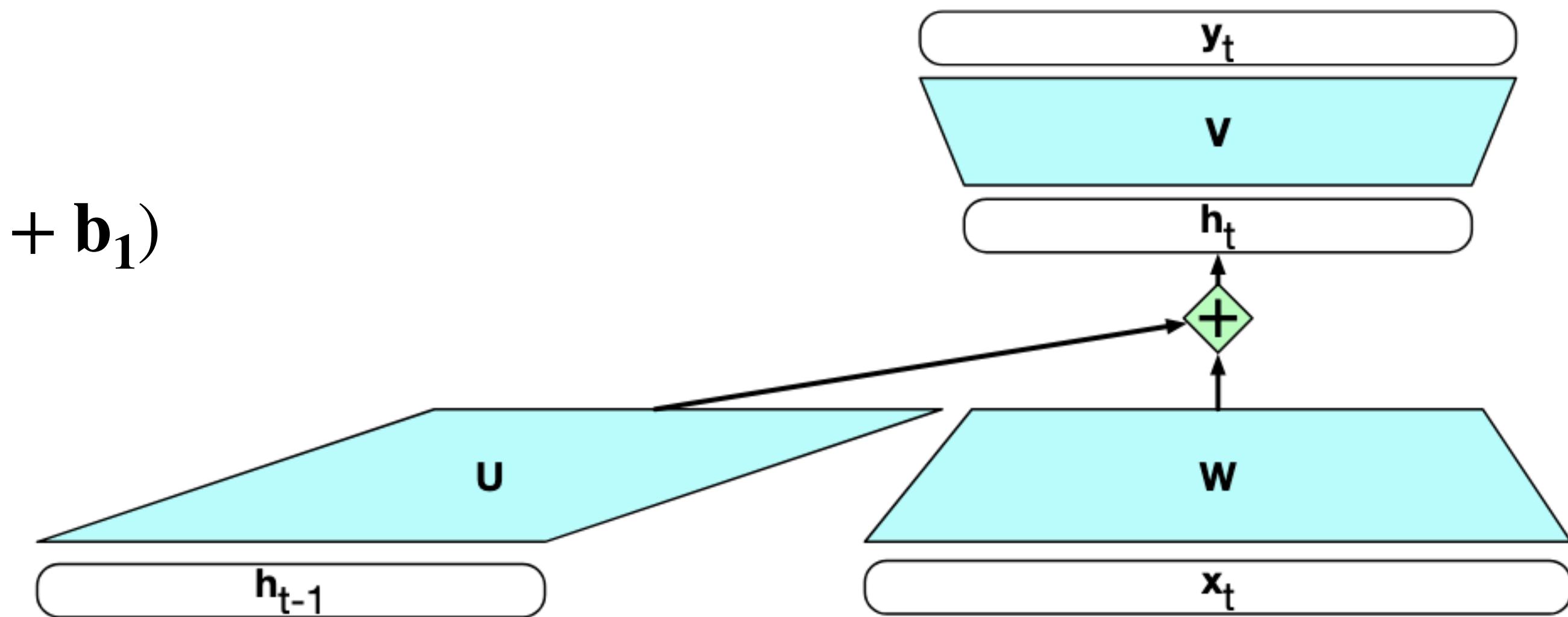
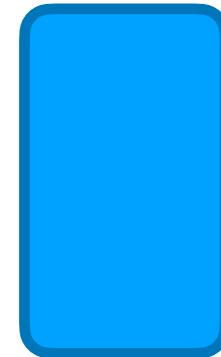


$$h = g(Wx_t + Uh_{t-1} + b_1)$$

**U**



= key new set of  
parameters to  
connect hidden

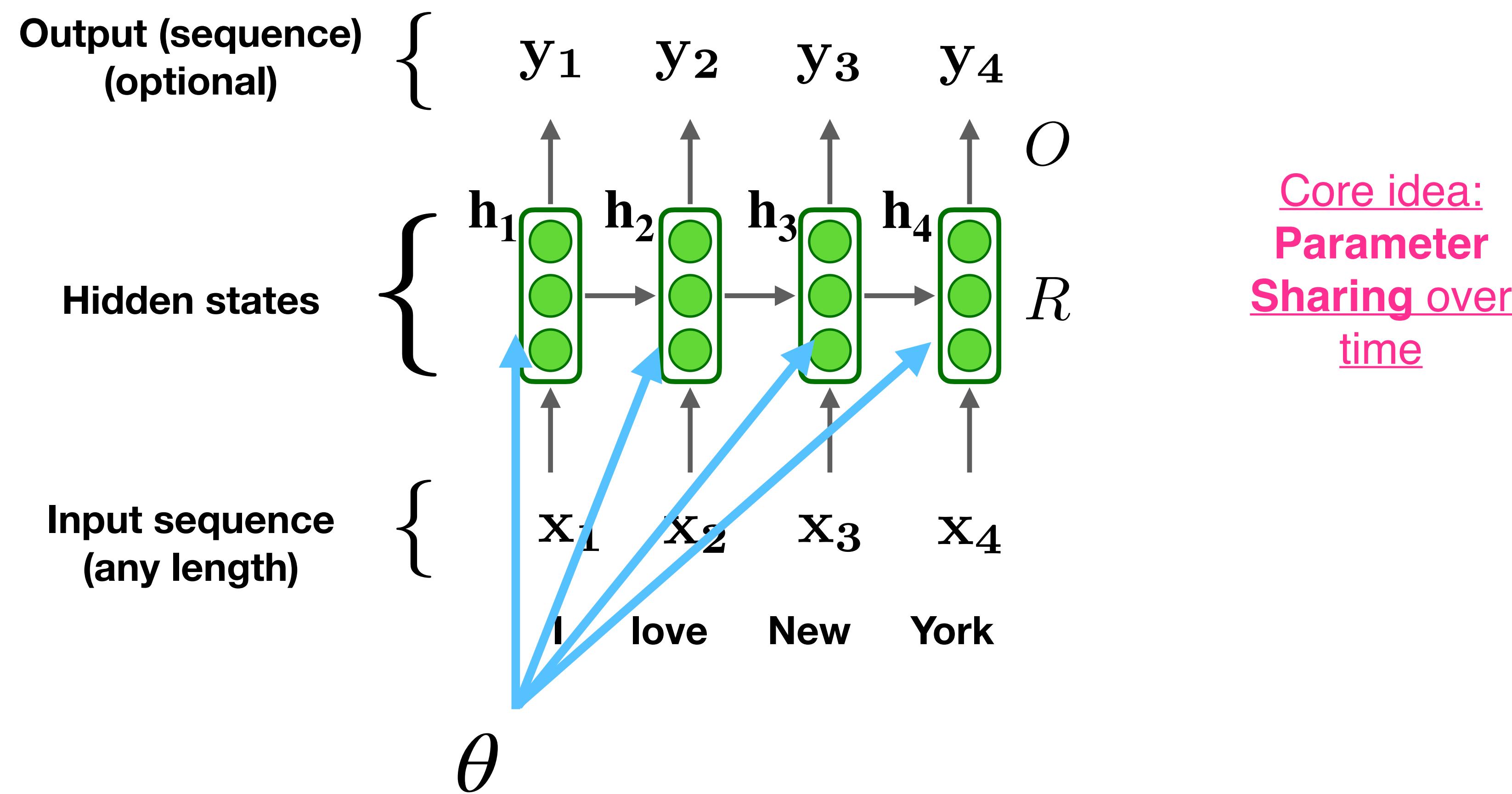


(Graphical illustration on right - from Jurafsky & Martin, SLP3)

# RNN - Step by Step

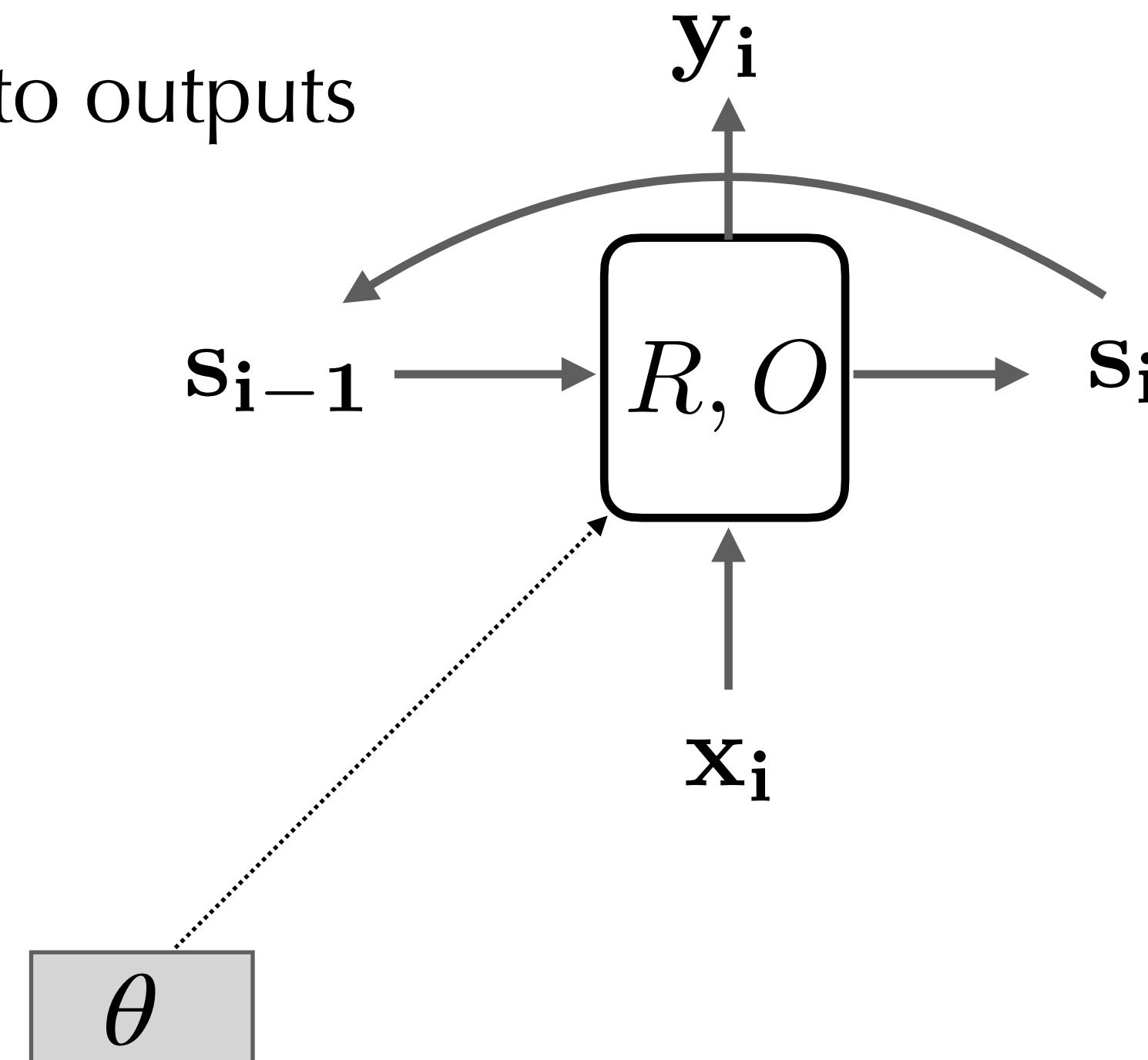
A family of recurrent NN architectures

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_1)$$



# The RNN abstraction

- ▶ Input sequence of vectors:  $\mathbf{x}_{1:n}$
- ▶  $RNN(\mathbf{s}_0, \mathbf{x}_{1:n})$  consists of two functions:
  - ▶ **Recurrence** function  $R$  consumes input and previous state
  - ▶ **Output** function  $O$  maps states to outputs
- ▶ Technically, there is a start state:  $\mathbf{s}_0$



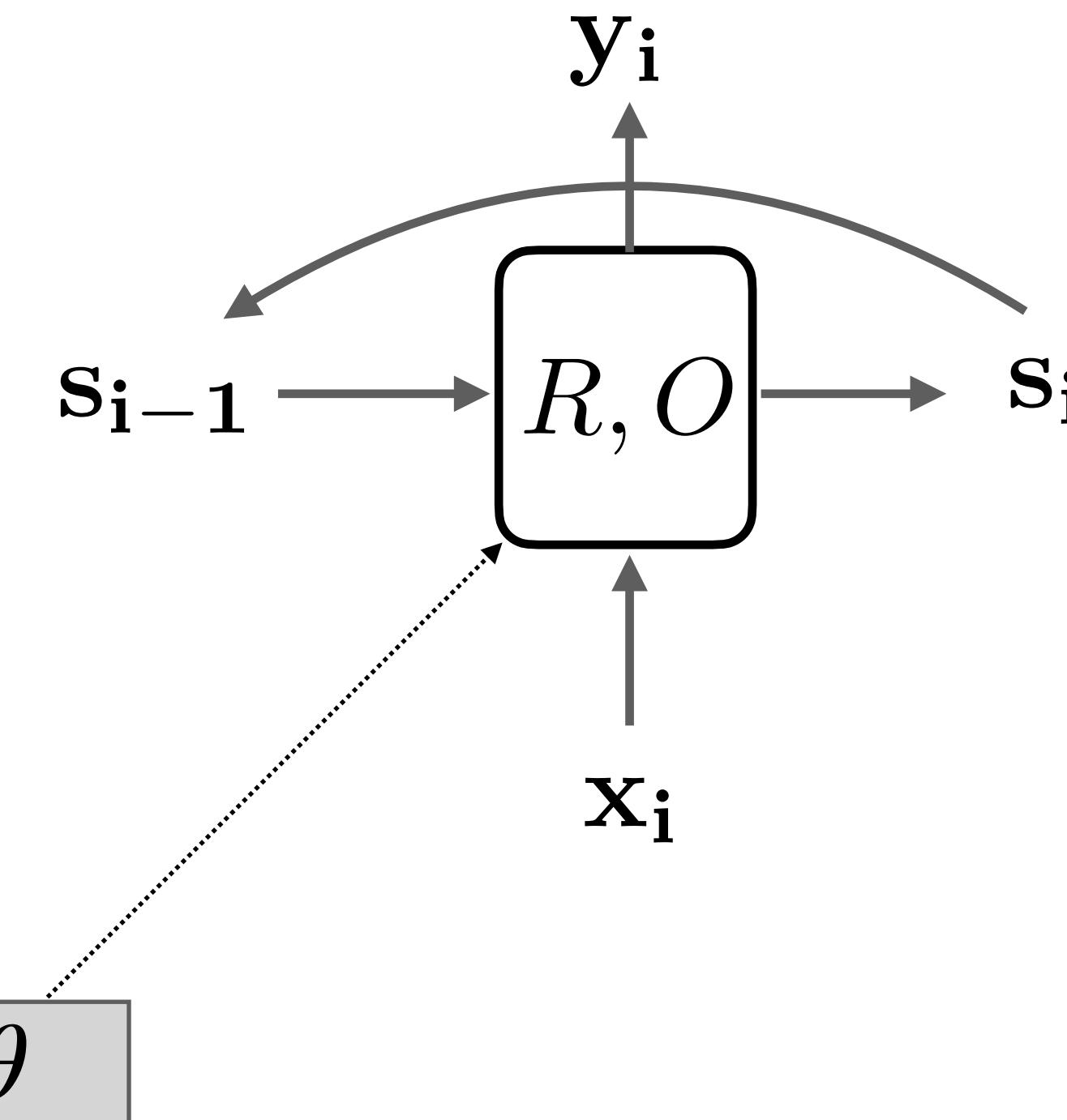
(Graphical illustration - from Yoav Goldberg's primer, 2015)

# The RNN abstraction - More formally

$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_{1:n}, \mathbf{y}_{1:n}$$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i)$$

$$\mathbf{y}_i = O(\mathbf{s}_i)$$



(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

# RNN: Unrolled over time

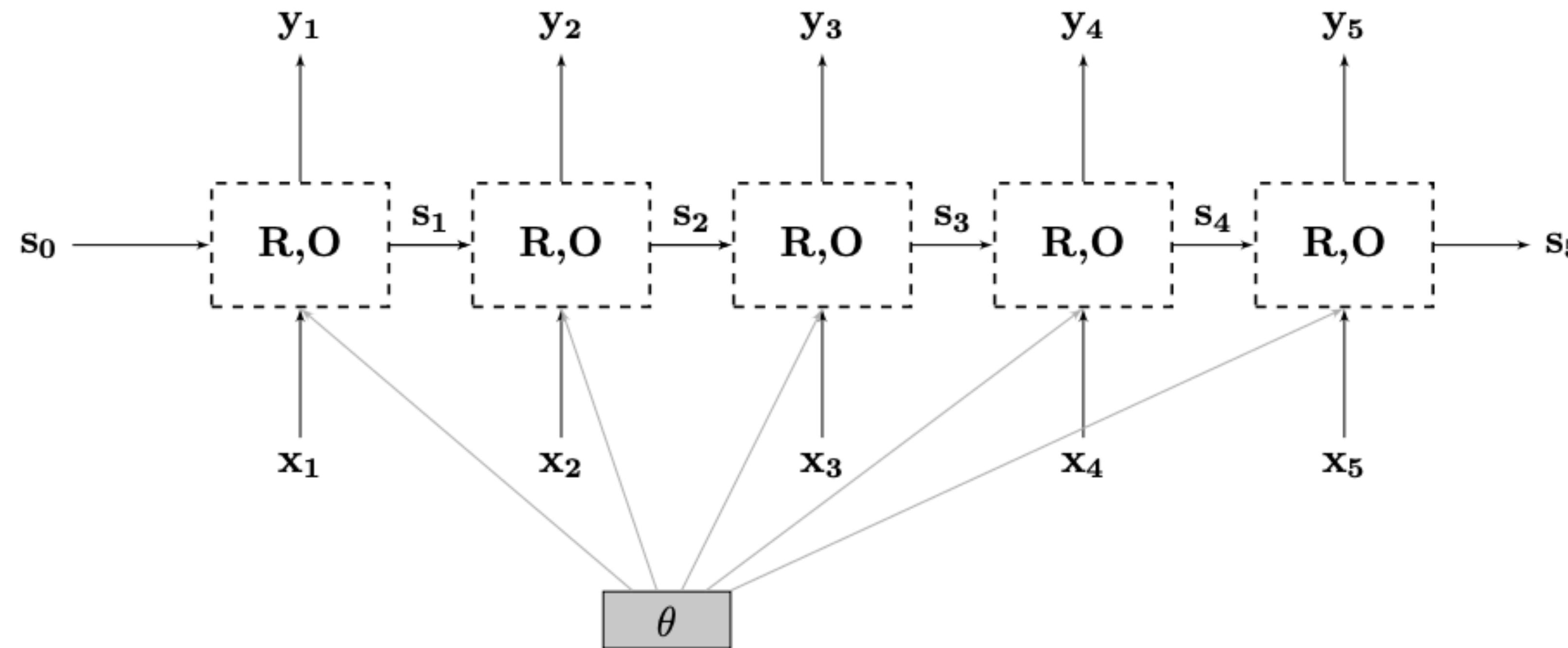


Figure 6: Graphical representation of an RNN (unrolled).

(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

# Expansion at time step 4

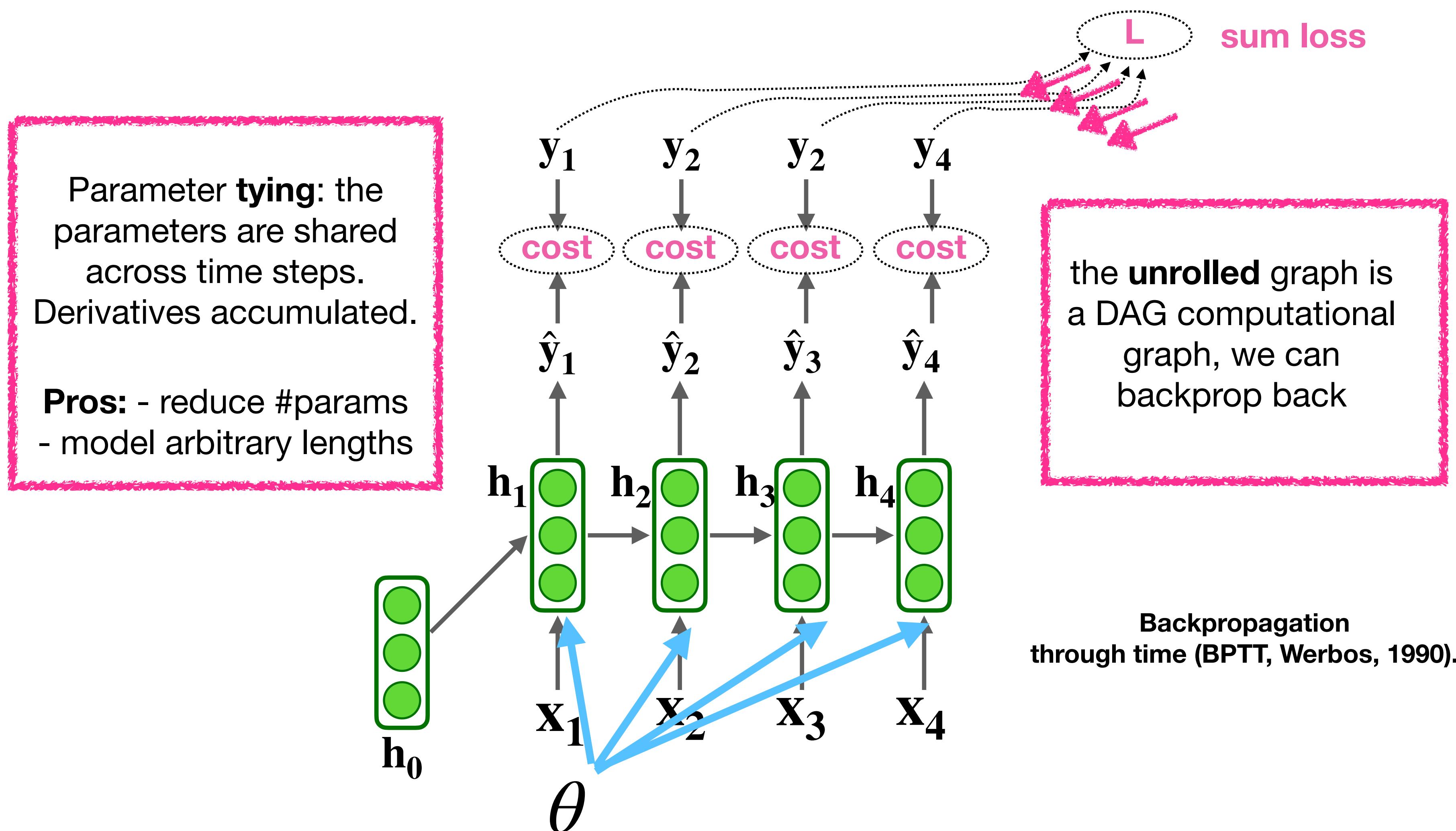
$$\mathbf{s}_4 = R(\mathbf{s}_3, \mathbf{x}_4)$$

$$= R(\overbrace{R(\mathbf{s}_2, \mathbf{x}_3)}^{\mathbf{s}_3}, \mathbf{x}_4)$$

$$= R(R(\overbrace{R(\mathbf{s}_1, \mathbf{x}_2)}^{\mathbf{s}_2}), \mathbf{x}_3), \mathbf{x}_4)$$

$$= R(R(R(\overbrace{R(\mathbf{s}_0, \mathbf{x}_1)}^{\mathbf{s}_1}), \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4)$$

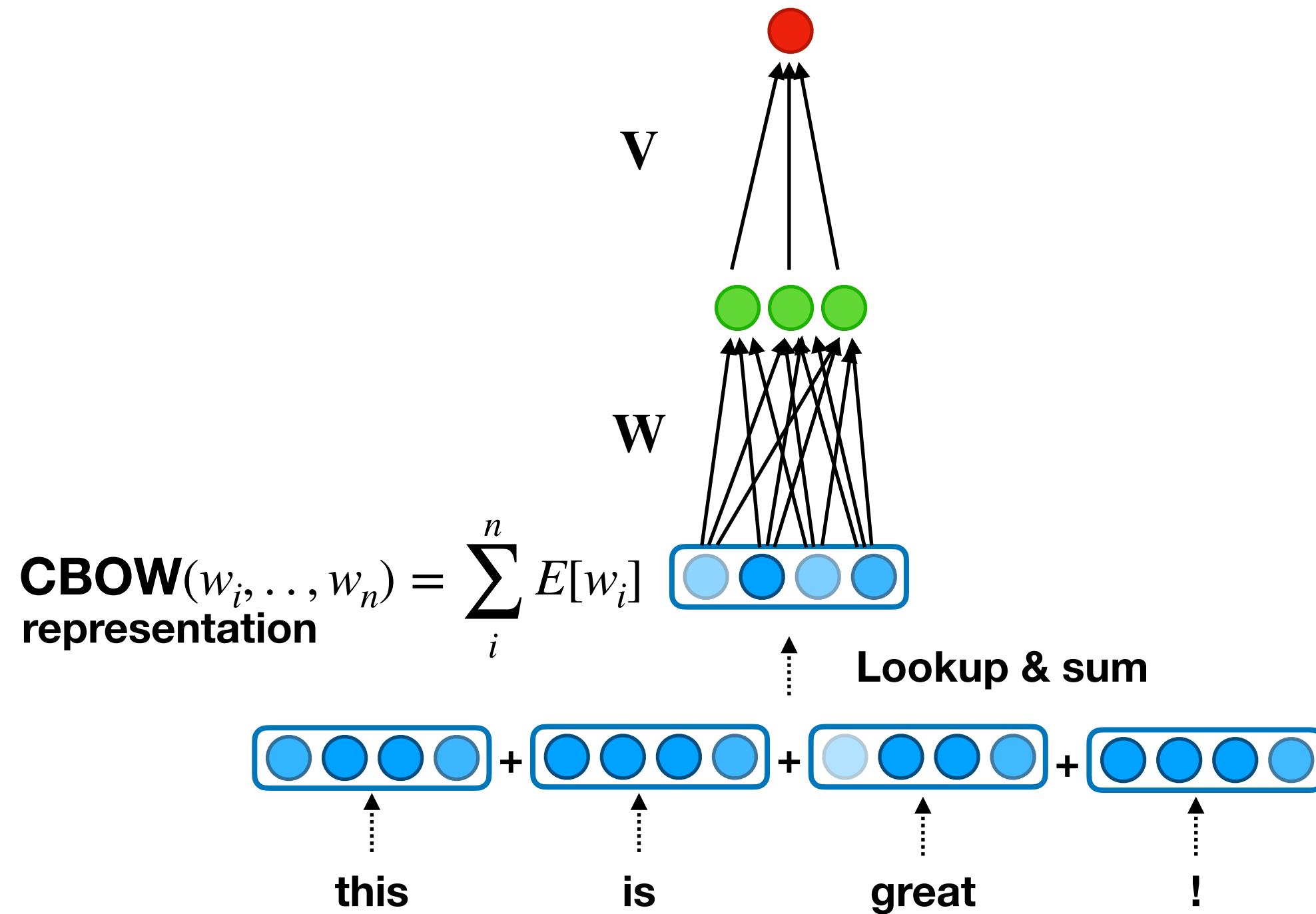
# Training a RNN, parameter tying



# Recap: FFNN's way

this is great !

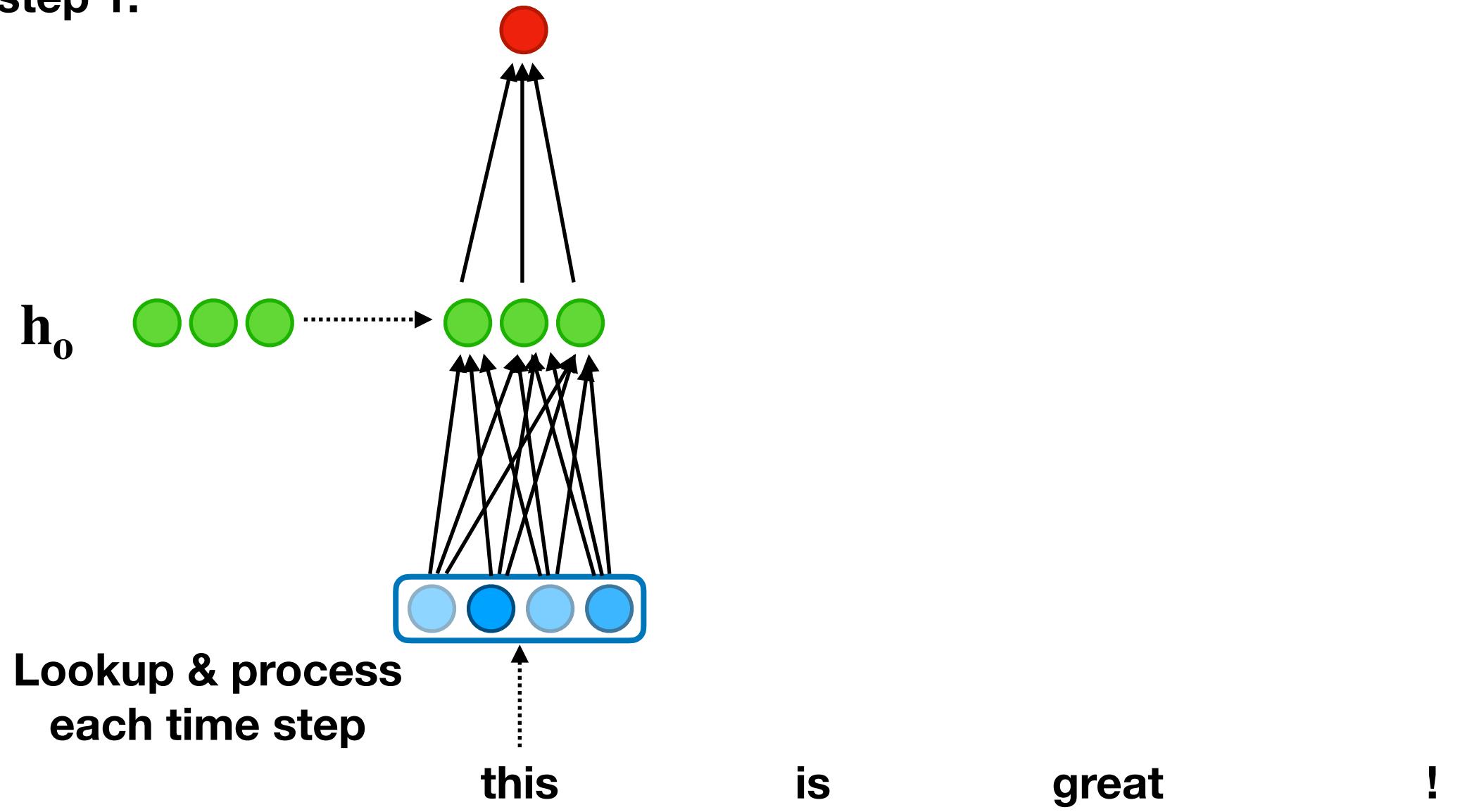
One fixed input, e.g. average or sum:



# Recap: RNN's way

this is great !

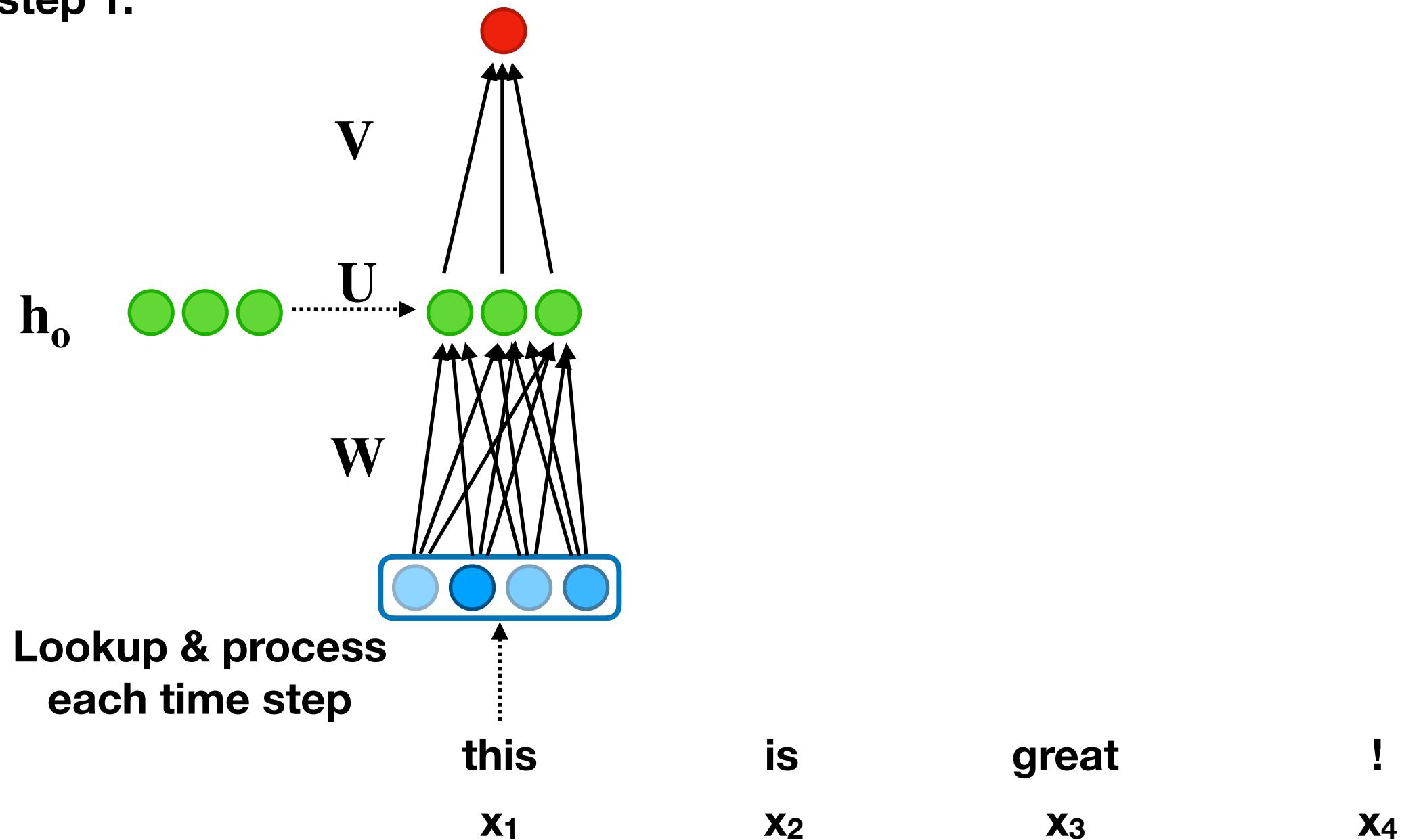
Time step 1:



# Recap: RNN's way

this is great !

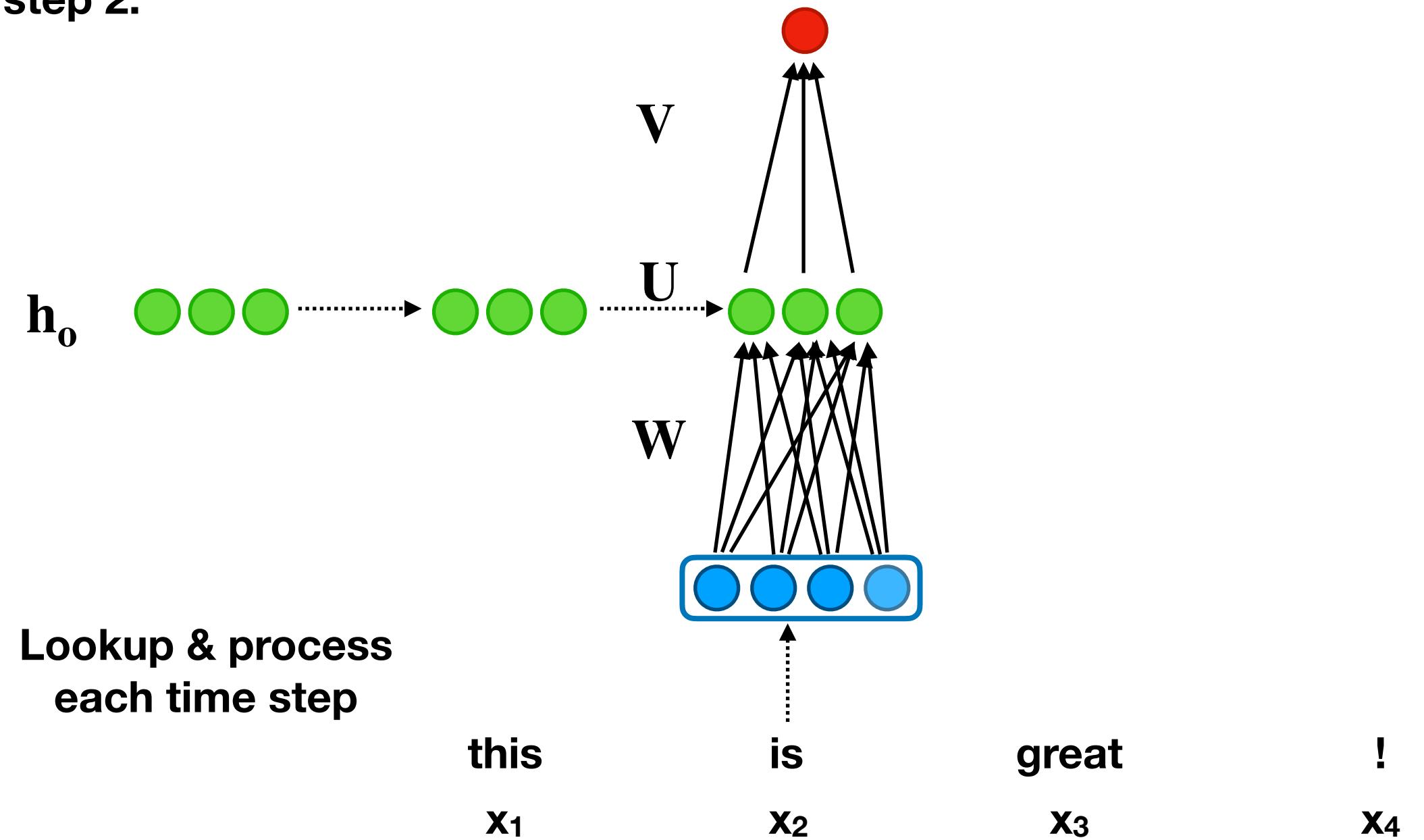
Time step 1:



# Recap: RNN's way

this is great !

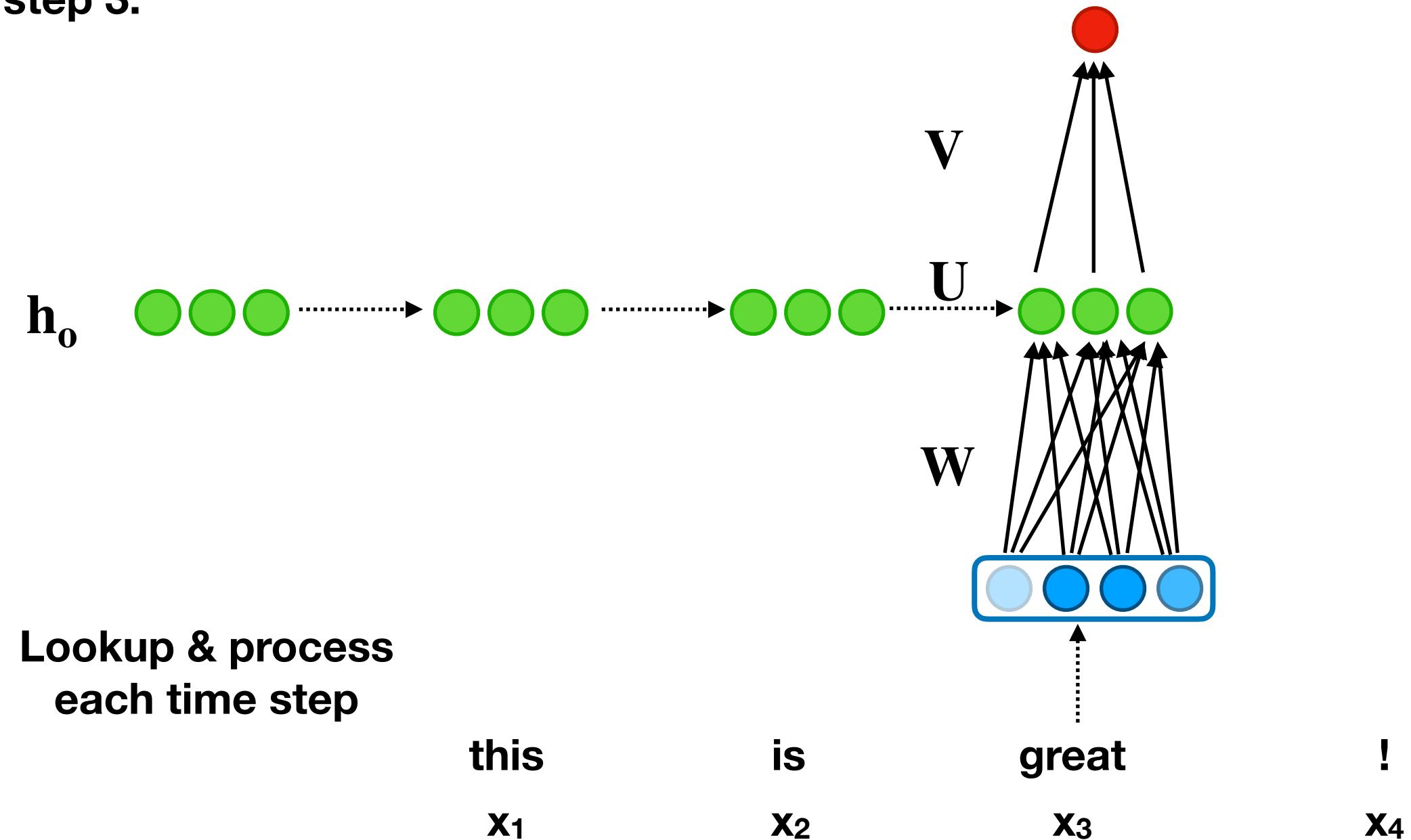
Time step 2:



# Recap: RNN's way

this is great !

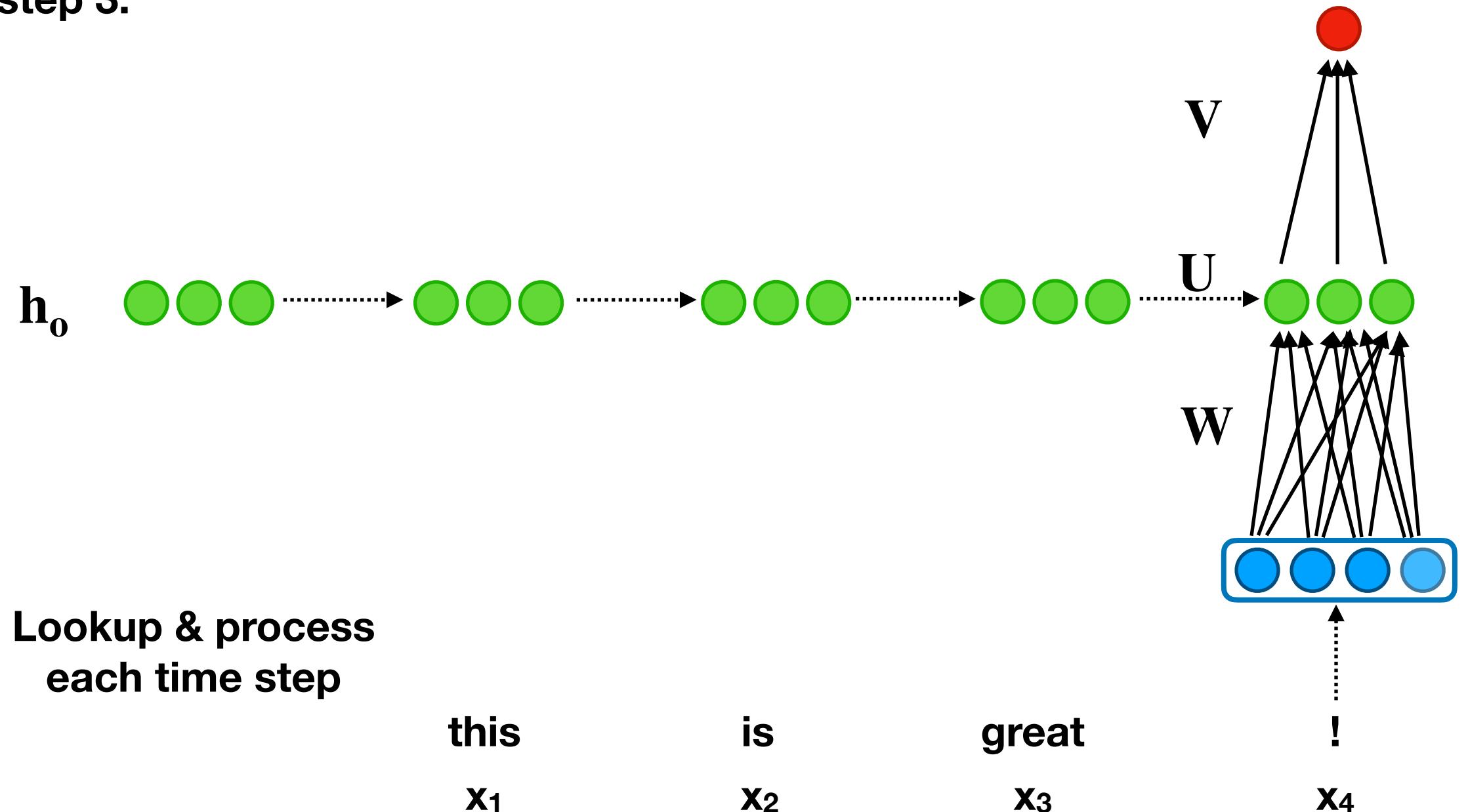
Time step 3:



# Recap: RNN's way

this is great !

Time step 3:

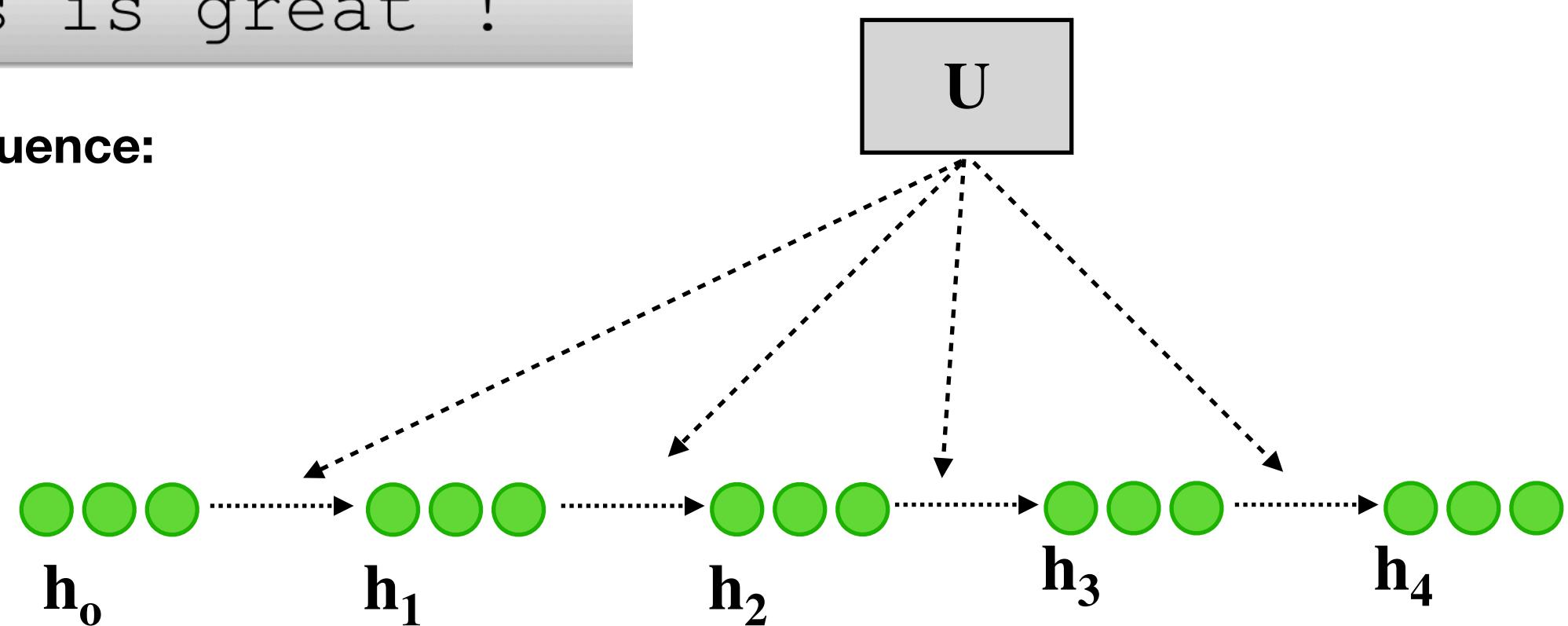


Lookup & process  
each time step

# RNN: shared parameters

this is great !

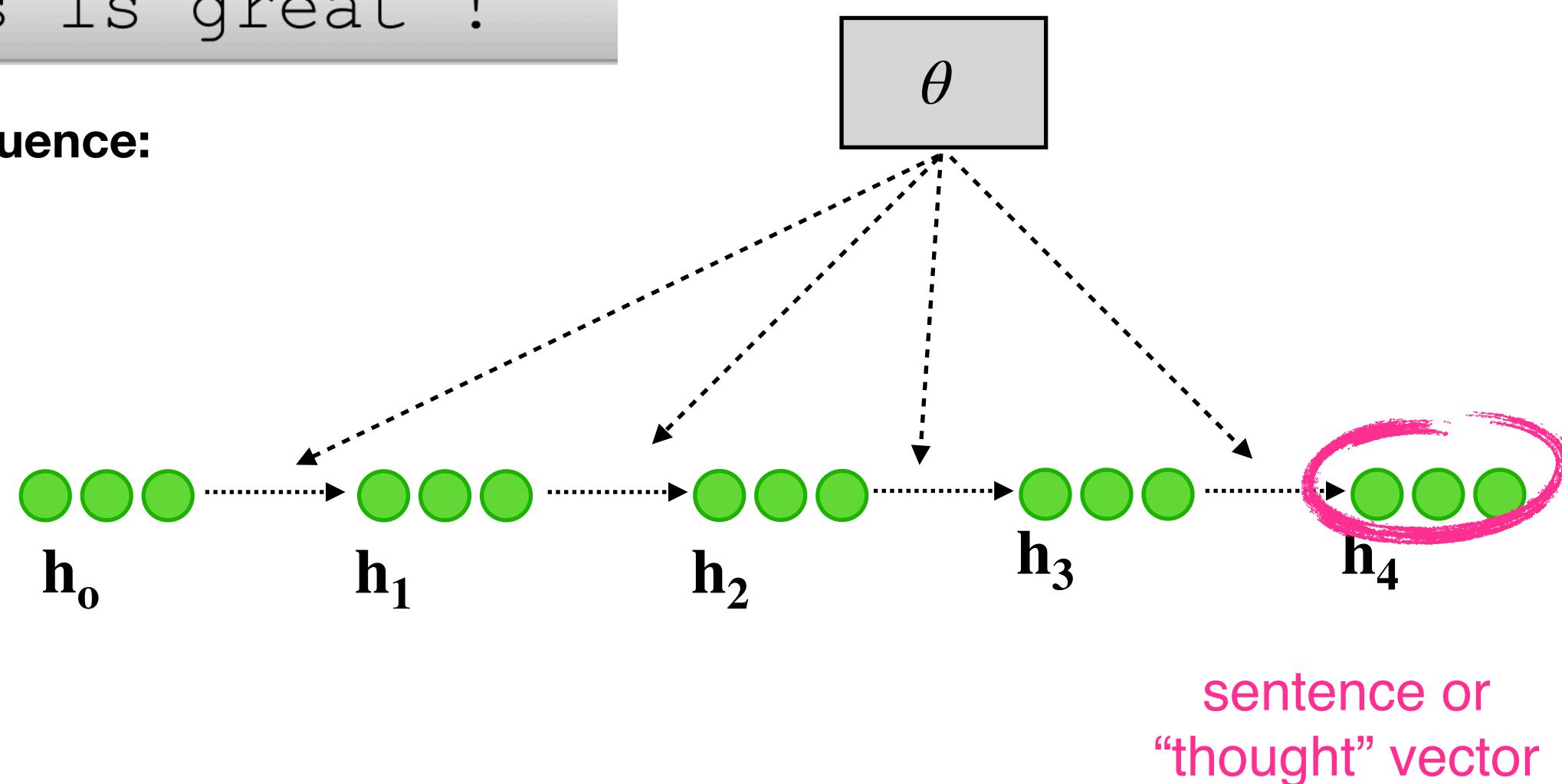
Unrolled sequence:



# RNN: $h$ acting as "memory"

this is great !

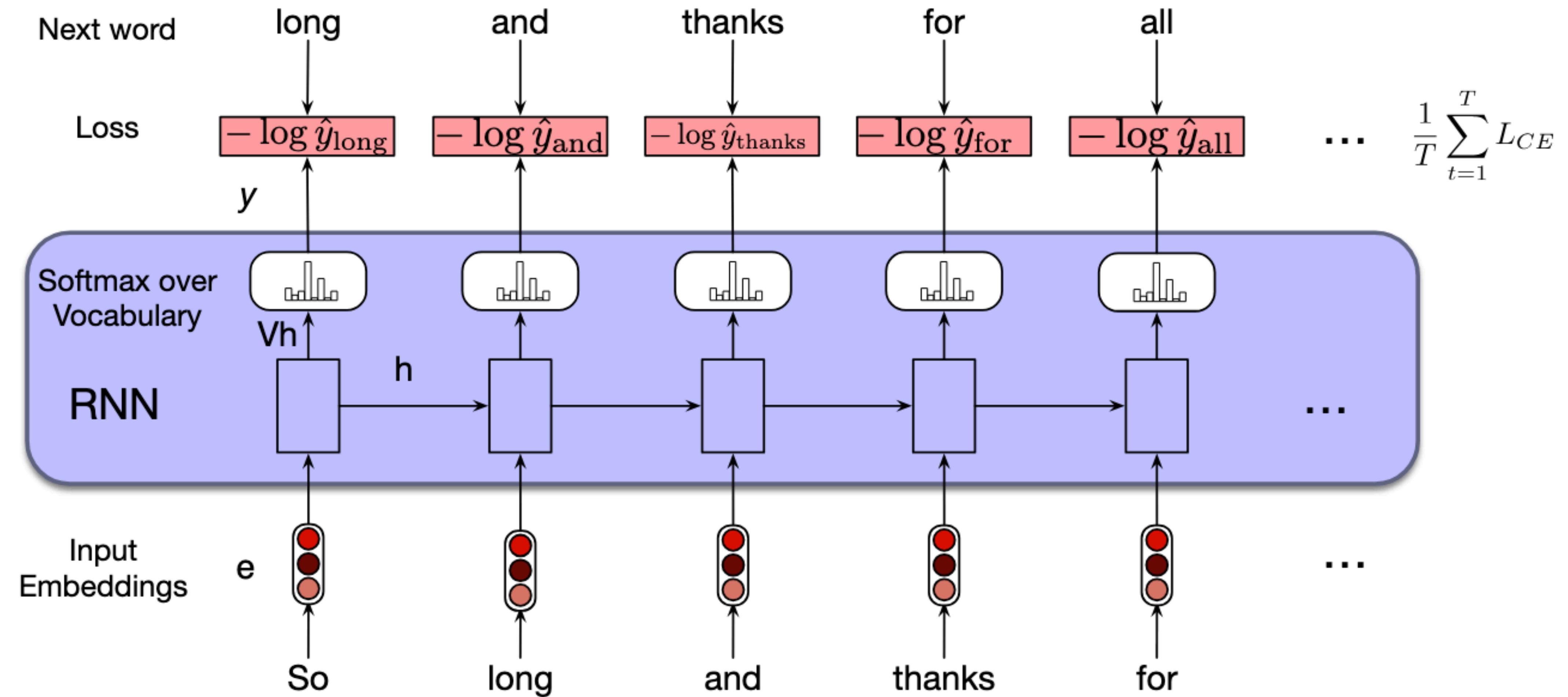
Unrolled sequence:



sentence or  
“thought” vector

# RNN Language Model

# Training a RNN LM - Example



# What about these issues?

- ▶ Can it handle similar words?
  - ▶ *she bought* a bicycle
  - ▶ *she purchased* a bicycle
- ▶ Long-distance dependencies?
  - ▶ *for programming* she yesterday purchased her own brand new *laptop*
  - ▶ *for running* she yesterday purchased her brand new *sportswatch*



However, in practice the vanilla RNN  
has some trouble.. more soon



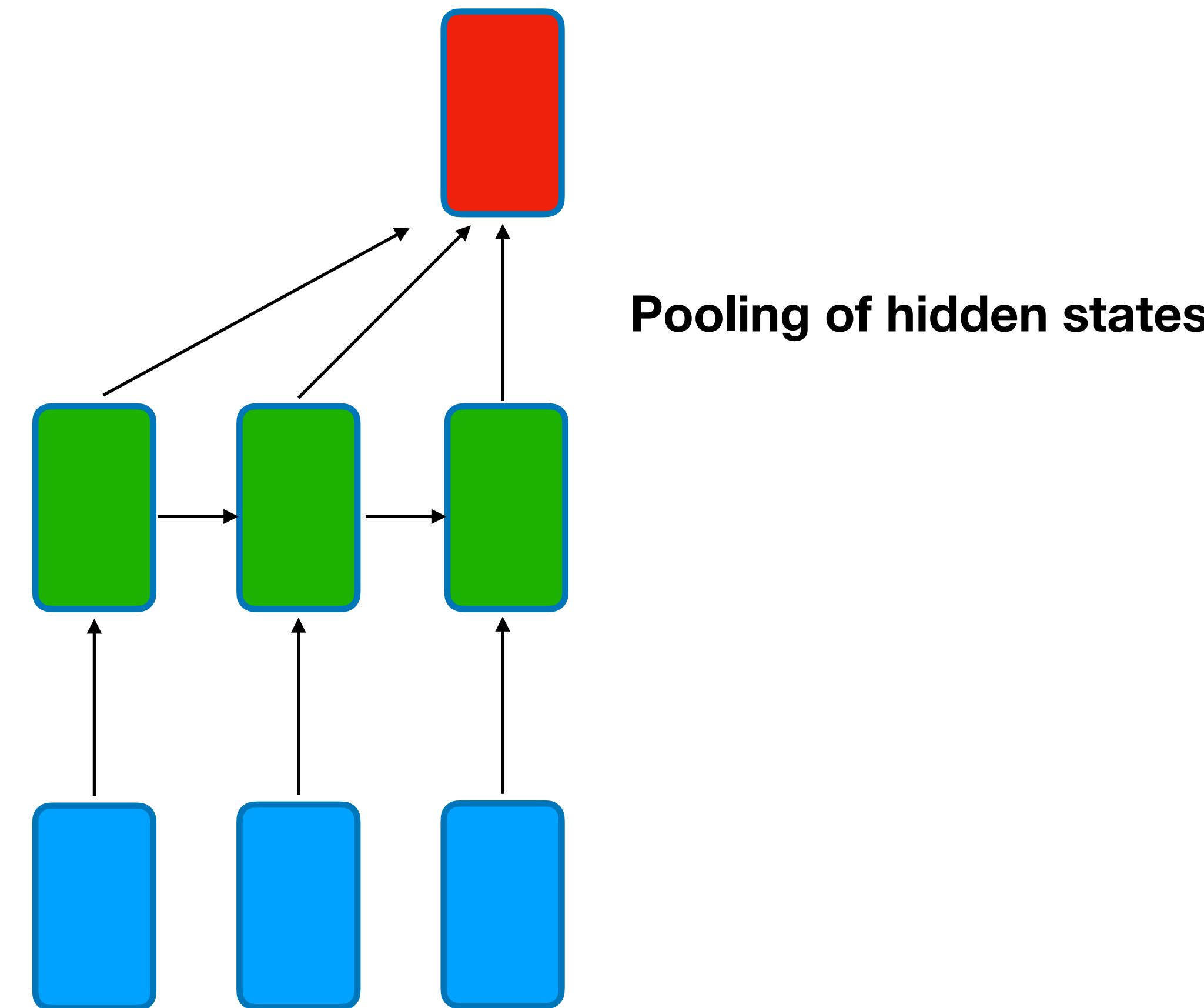
# RNNs - Interim summary

- ▶ **LM:** a model that predicts the next word
- ▶ **RNN:** a family of neural networks
  - ▶ to model sequential input of any length
  - ▶ can optionally produce an output at each time step  $t$  (function  $O$ )
- ▶ RNNs are great as LMs. They can be used for much more:

# **Four Common Usage Patterns of RNNs**

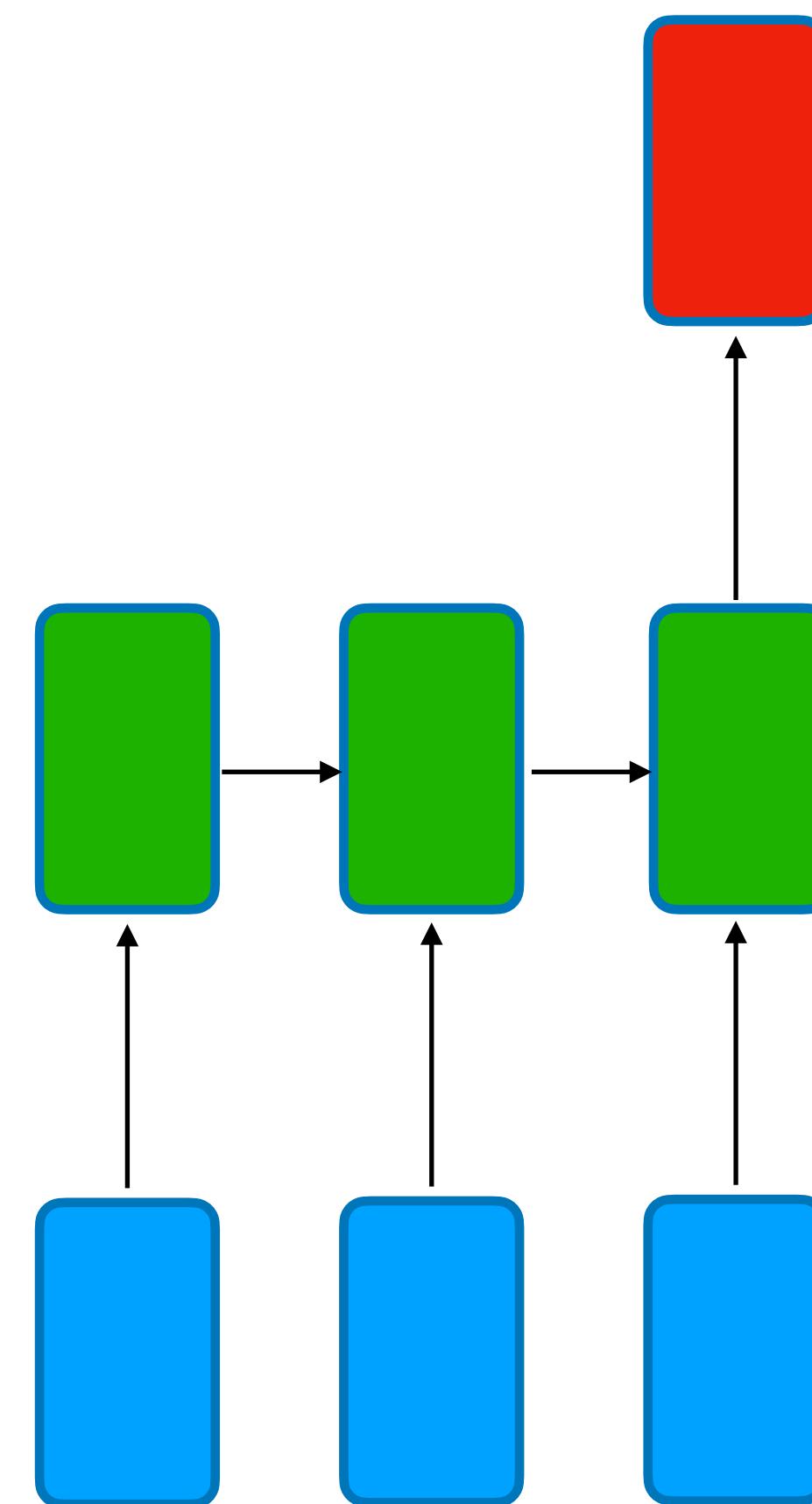
# An RNN as acceptor

- ▶ Use **average of states** to predict  $y$
- ▶ Use **last state only** to predict  $y$



# Example: An RNN as encoder

- ▶ Use **last state** as encoding of the information in the sequence; use as “feature” in other NN
  - ▶ encode, not predict
  - ▶ E.g. character RNN



# RNN as Transducer

- ▶ predict an output for each time step  $t$
- ▶ E.g. Tagging (POS, NER)

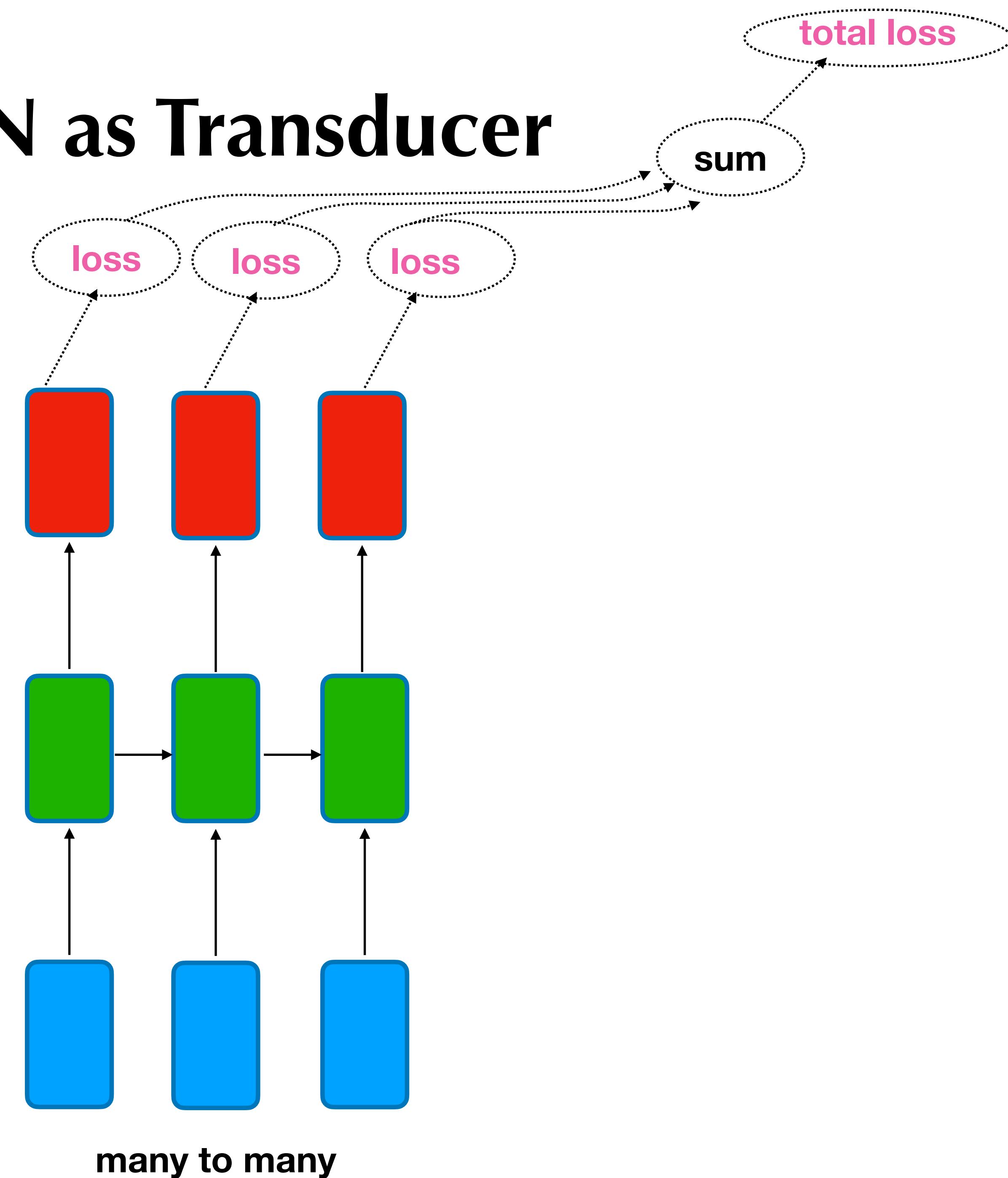
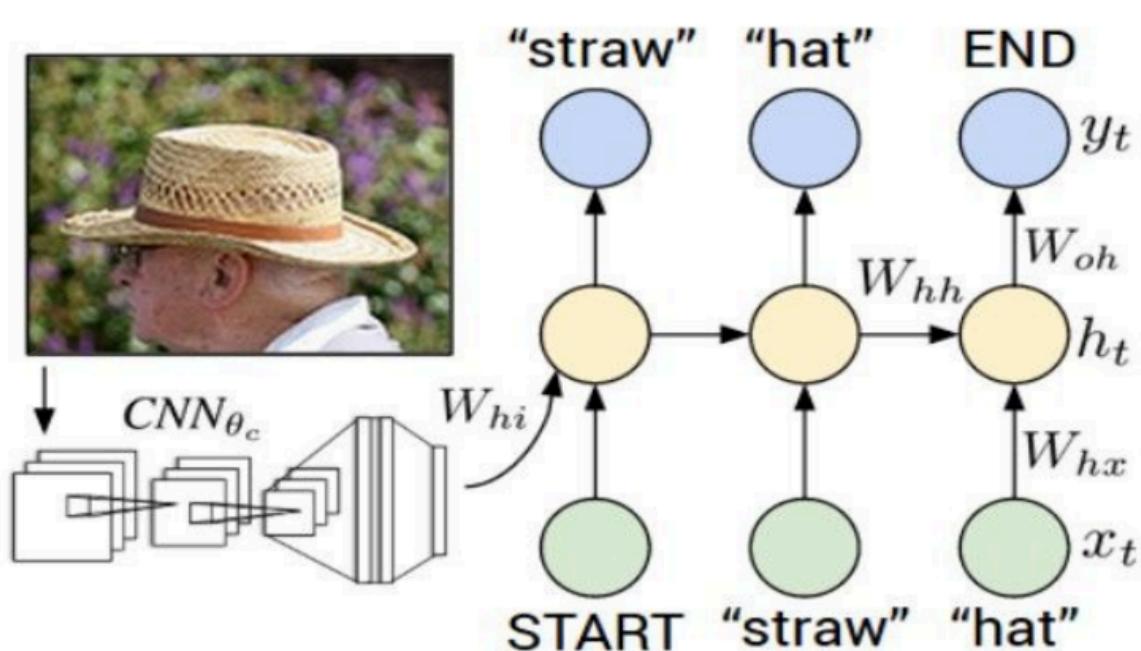
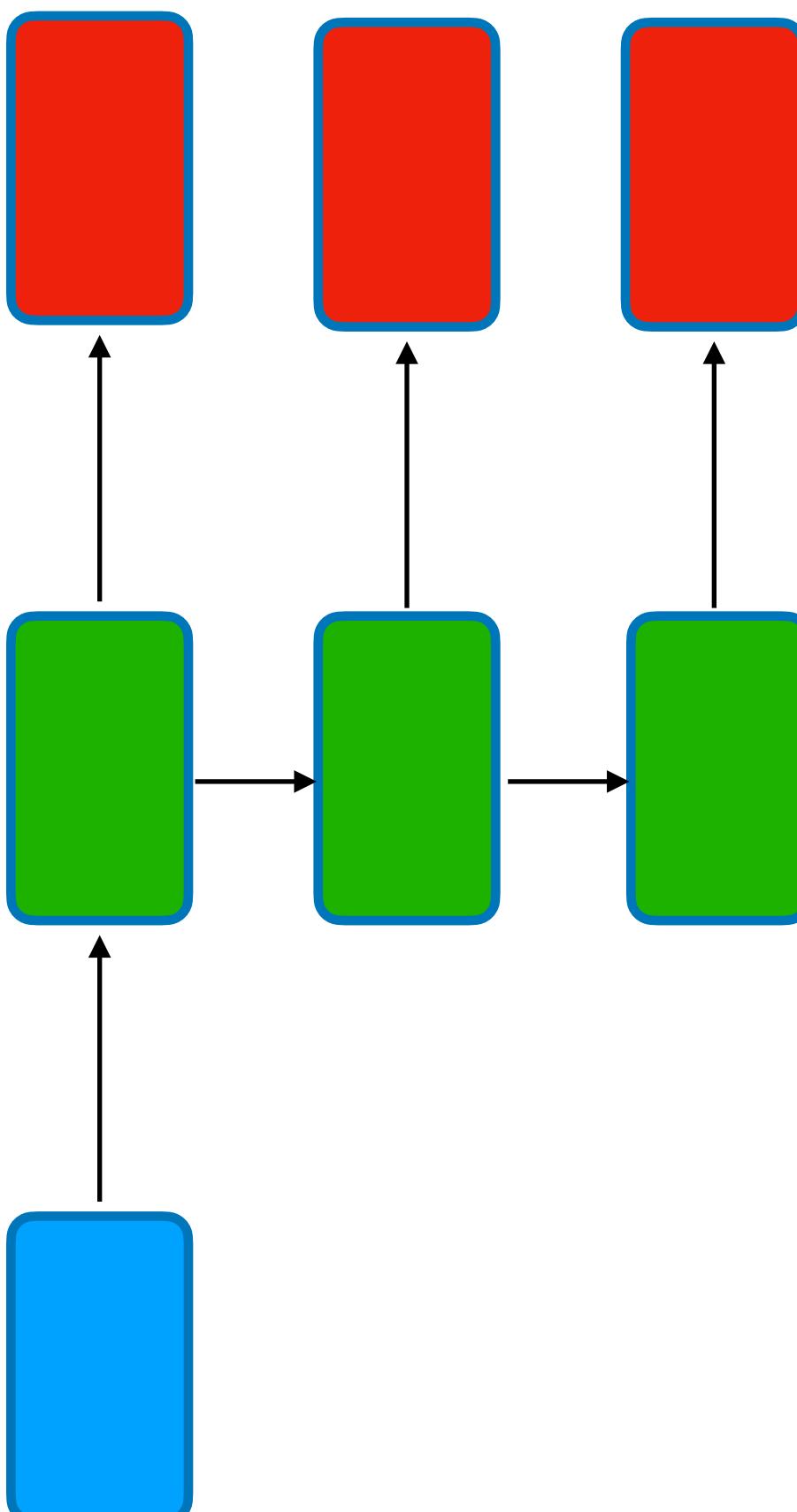


Illustration adapted from Karpathy

# RNN as generator

- ▶ Conditioned generation
- ▶ E.g. image caption generation, speech synthesis

**one to many**

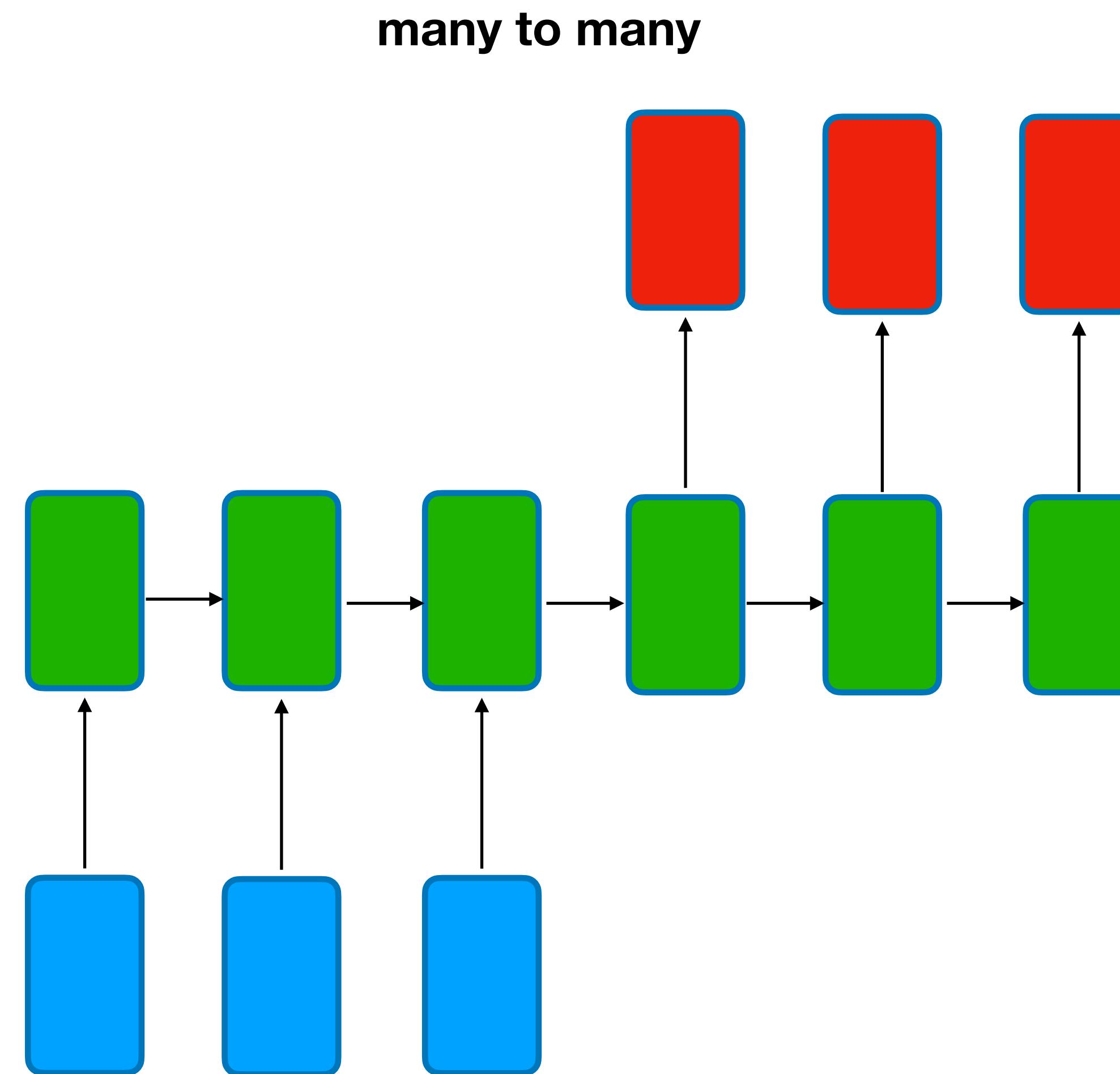


Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Illustration adapted from Karpathy

# RNN encoder-decoder (seq2seq)

- Both input and output are a sequence

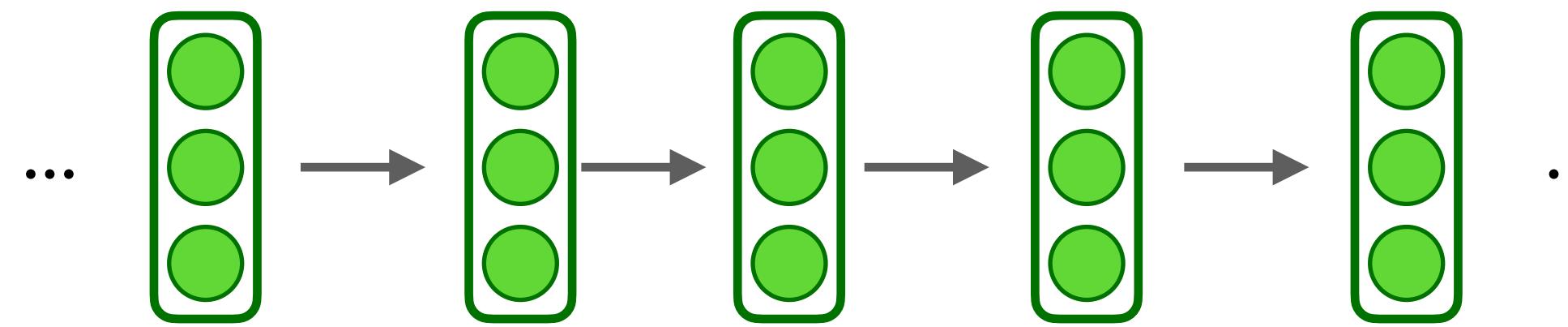


**Deeper, better models?  
Some more concepts**

# Only left to right?

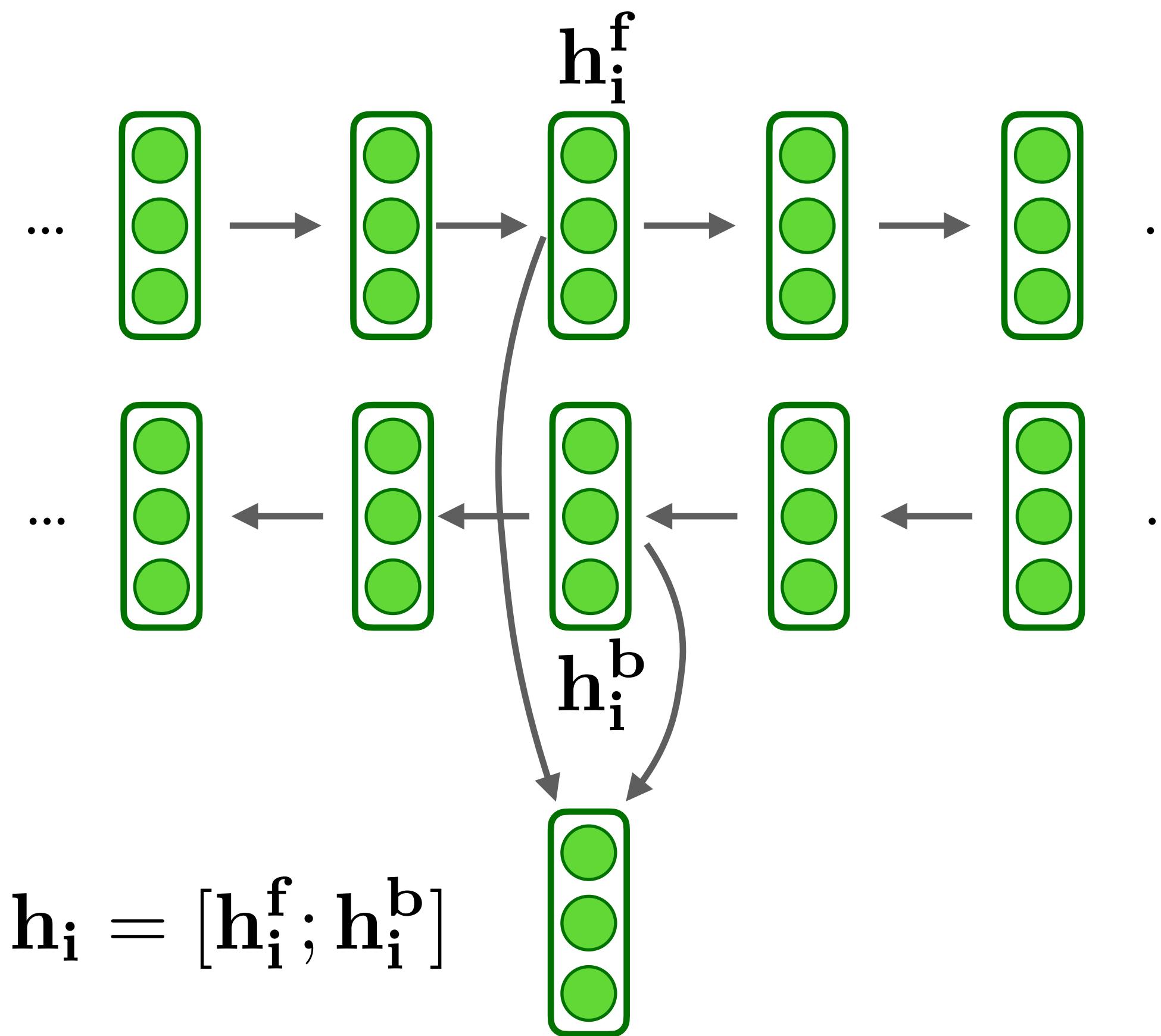
The person who hunts ducks out on the weekends

... person who hunts ducks out ...



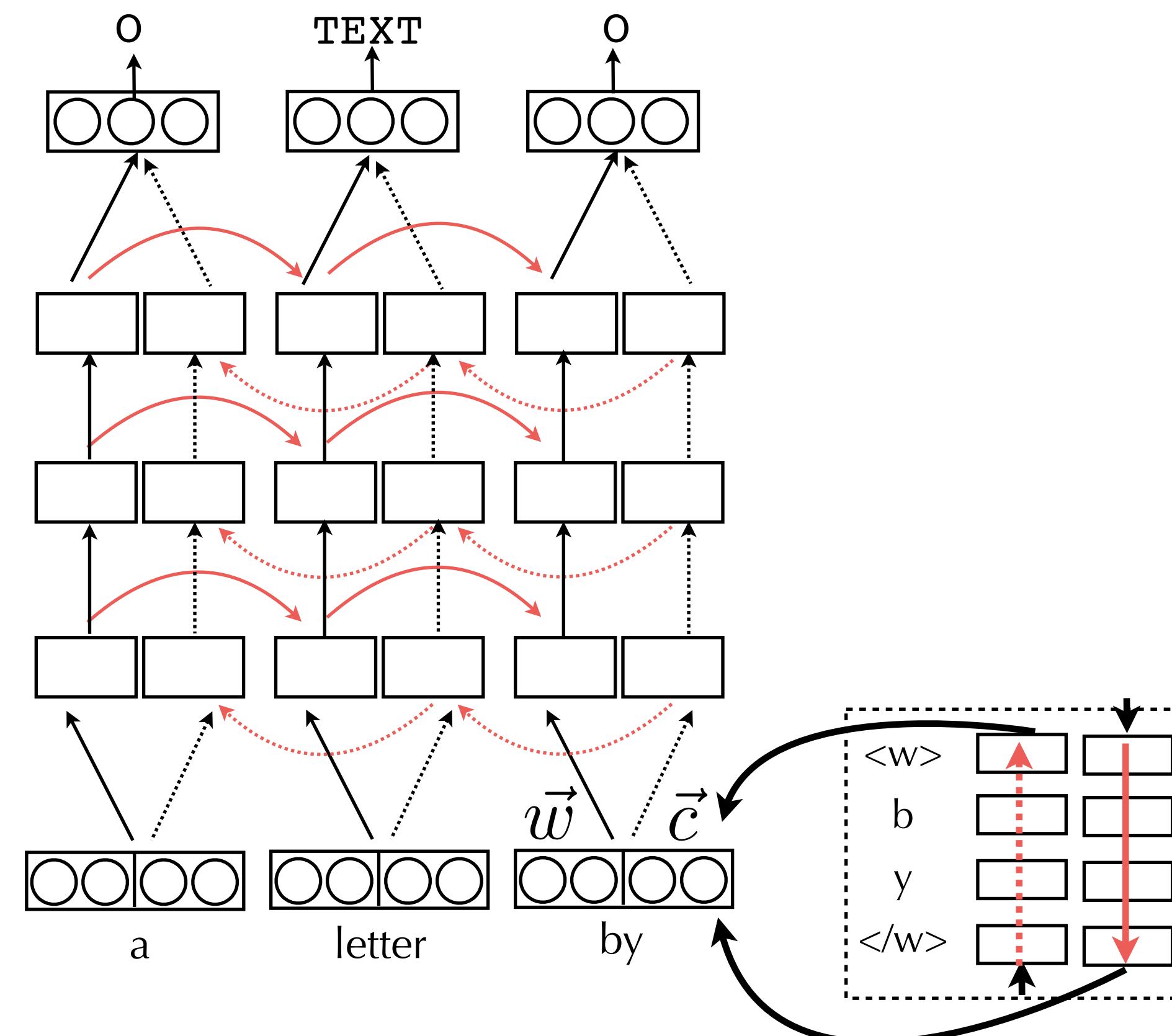
# Bidirectional RNNs

... person who hunts ducks out ...

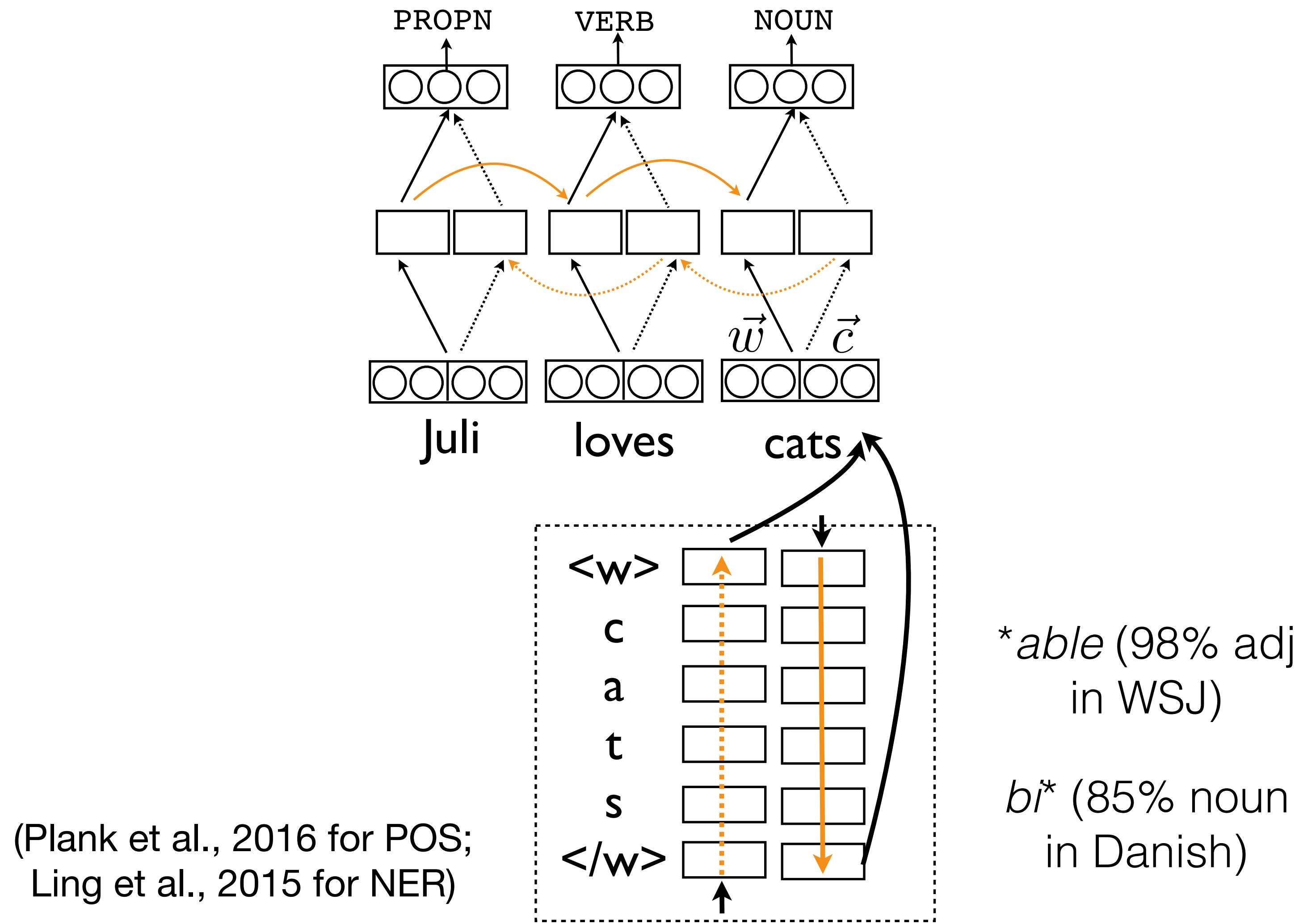


# Stacked RNNs

- Multiple layers of RNNs, e.g., bi-RNNs

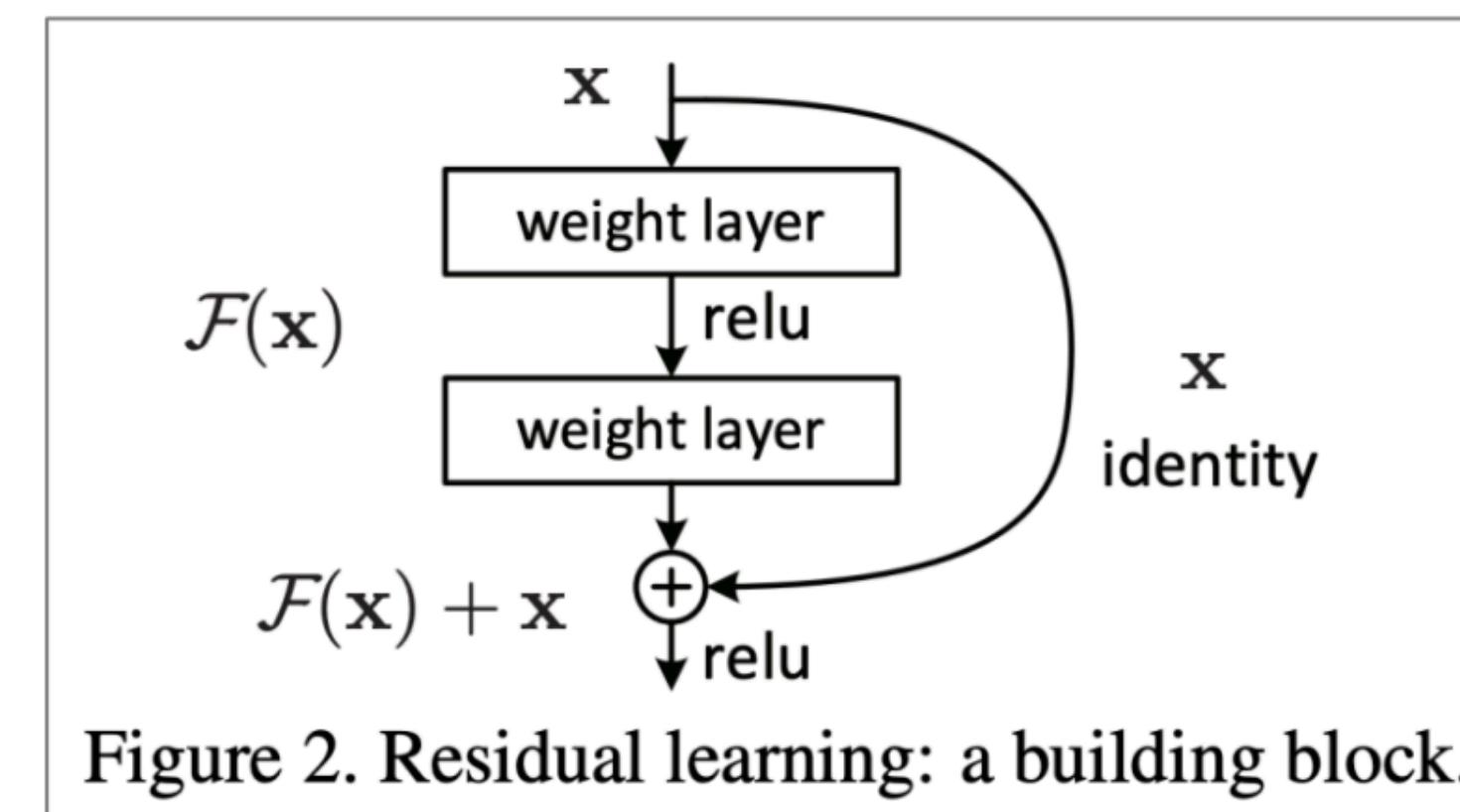


# Subword representations: Characters



# Residual connections

- ▶ Training deep neural networks is difficult
- ▶ **Solution:** add direct “skip” connections (ResNet, residual connections), proposed by He et al., (2015)
  - ▶ i.e. add  $F(x) + x$ , instead of  $F(x)$
  - ▶ allows for training deeper models (in fact, is used in transformers as we will see)



# Gated RNN architectures

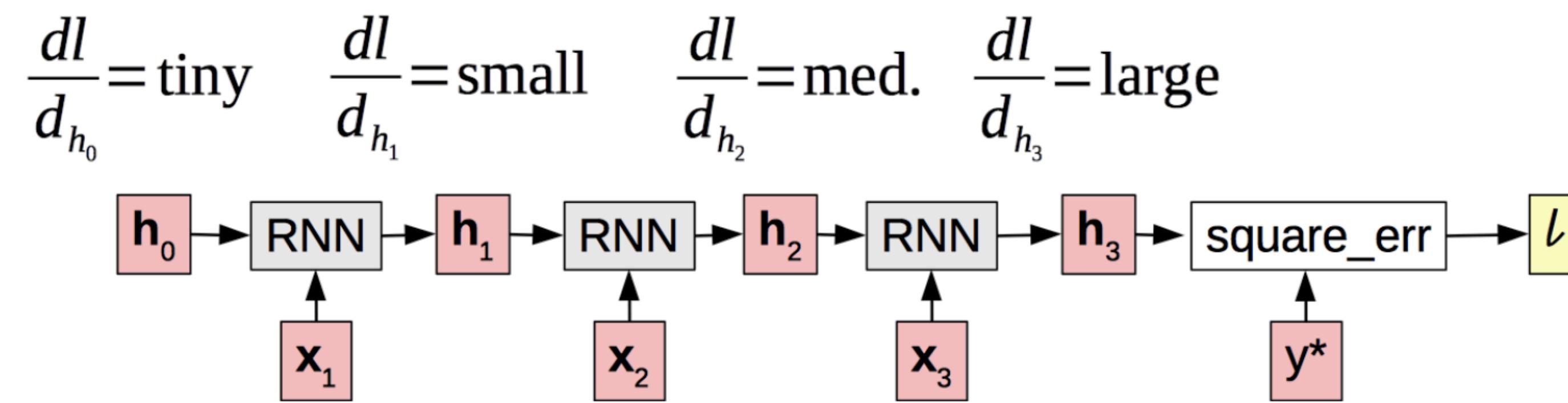
# A note on terminology

- ▶ RNN = “vanilla” RNN
- ▶ RNN flavors (=gated RNNs):
  - ▶ GRU  and LSTMs 
  - ▶ Why? Problem of RNNs: Vanishing gradients!



# Vanishing Gradient

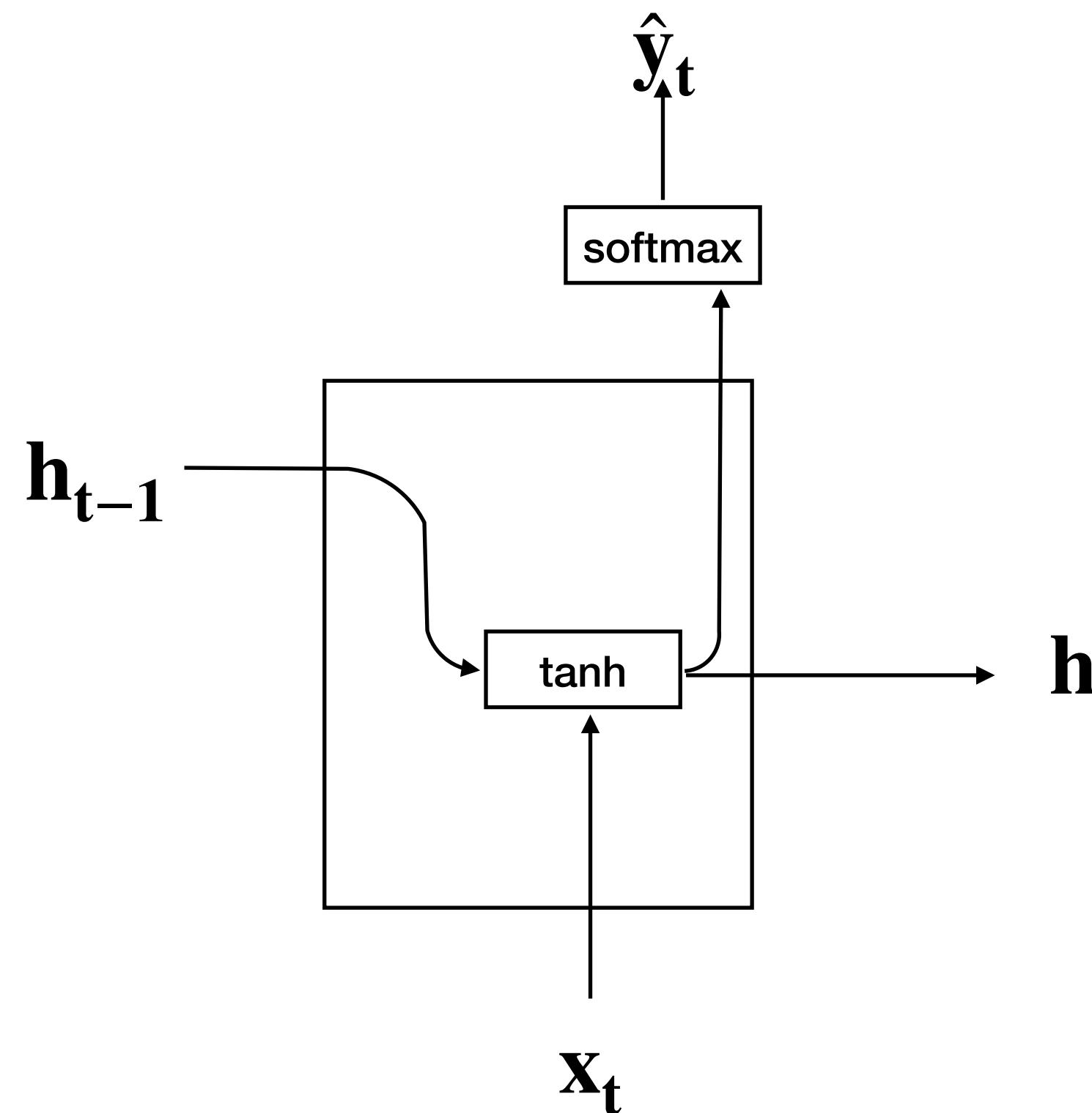
- Gradients decrease as they are pushed back



(Illustration by Graham Neubig)

**Gated RNN architectures:  
RNN flavors with a  
separate memory**

# Vanilla RNN unit



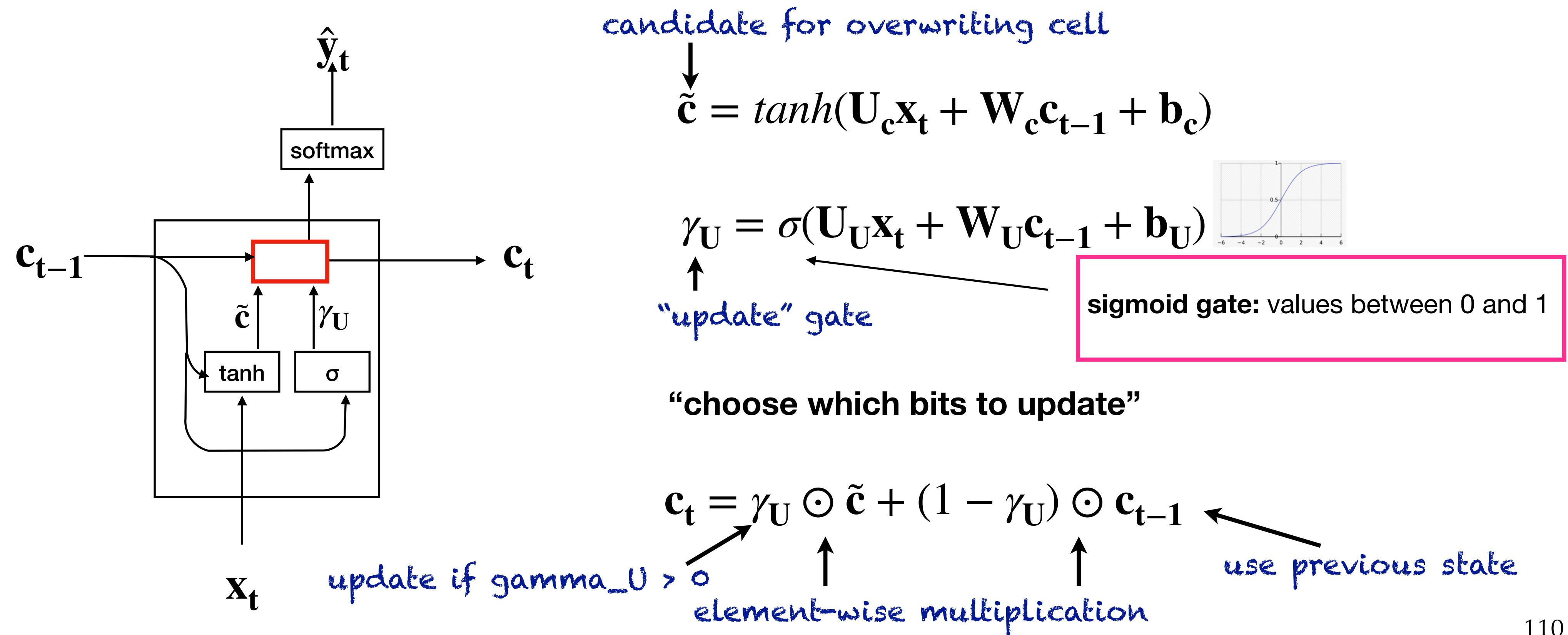
At each time step, the hidden state is updated:

$$h_t = g(Wx_t + Uh_{t-1} + b)$$

in a vanilla RNN  
the hidden state is  
constantly being  
rewritten

# GRU (Gated recurrent Unit) - simplified

- Cho et al. (2014) - key idea: dynamic memory update  $\mathbf{c}$  ( $h=c$ )
- at every step  $t$ , consider overwriting candidate memory



# GRU (Gated recurrent Unit) - full

- GRU: the full GRU has two gates:

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

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

$$\begin{aligned}\tilde{\mathbf{c}} &= \tanh(\mathbf{U}_c \mathbf{x}_t + \mathbf{W}_c (\gamma_R \odot \mathbf{c}_{t-1}) + \mathbf{b}_c) \\ \gamma_U &= \sigma(\mathbf{U}_U \mathbf{x}_t + \mathbf{W}_U \mathbf{c}_{t-1} + \mathbf{b}_U) \\ \gamma_R &= \sigma(\mathbf{U}_R \mathbf{x}_t + \mathbf{W}_R \mathbf{c}_{t-1} + \mathbf{b}_R) \\ \mathbf{c}_t &= \gamma_U \odot \tilde{\mathbf{c}} + (1 - \gamma_U) \odot \mathbf{c}_{t-1}\end{aligned}$$

all vectors of same size

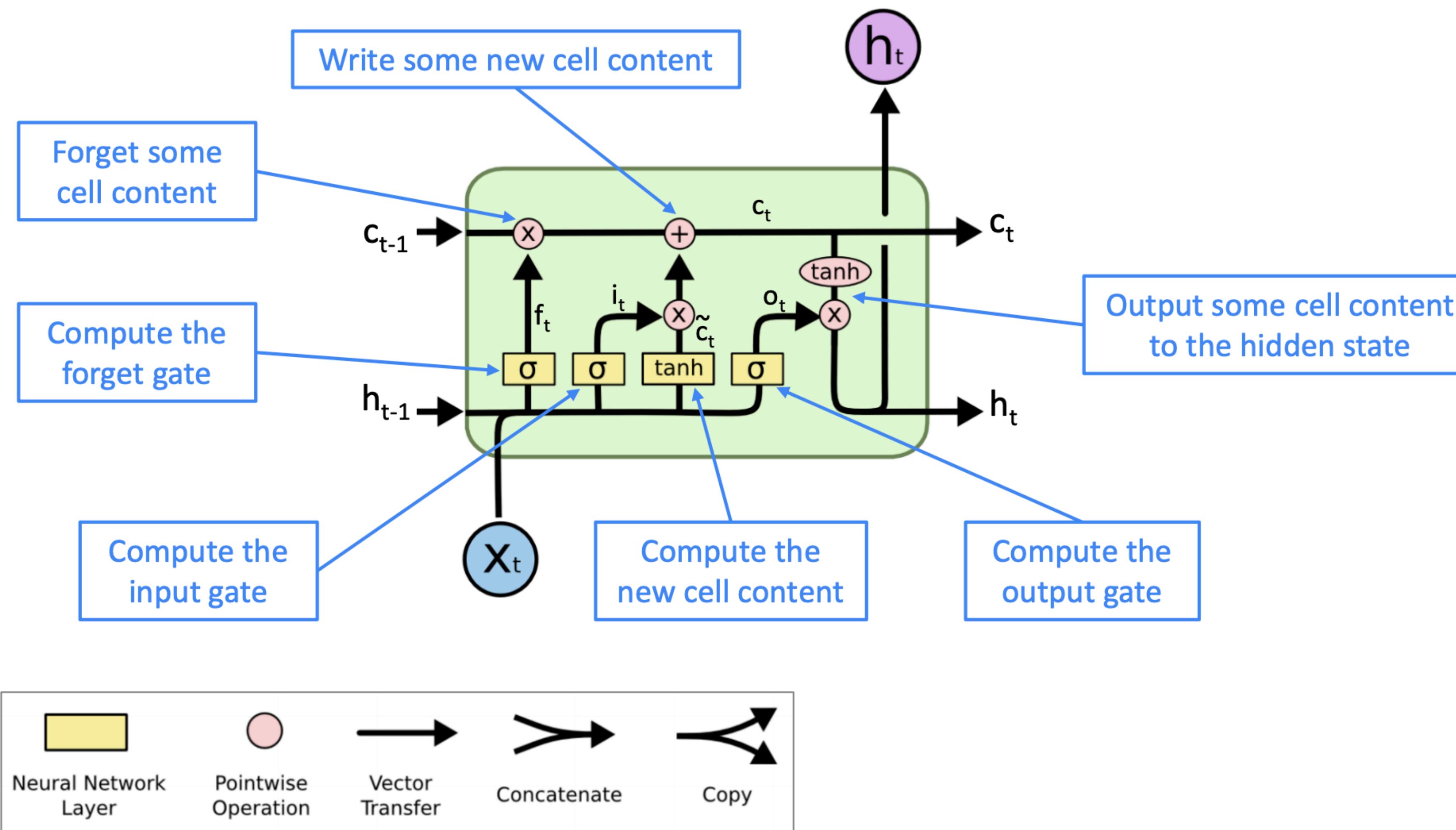
**How does this help the vanishing gradient problem?**  
GRUs make it easier to retain info long-term (e.g. by not updating bits)

Slide inspired by Abigail See

# LSTM (Long-Short Term Memory)

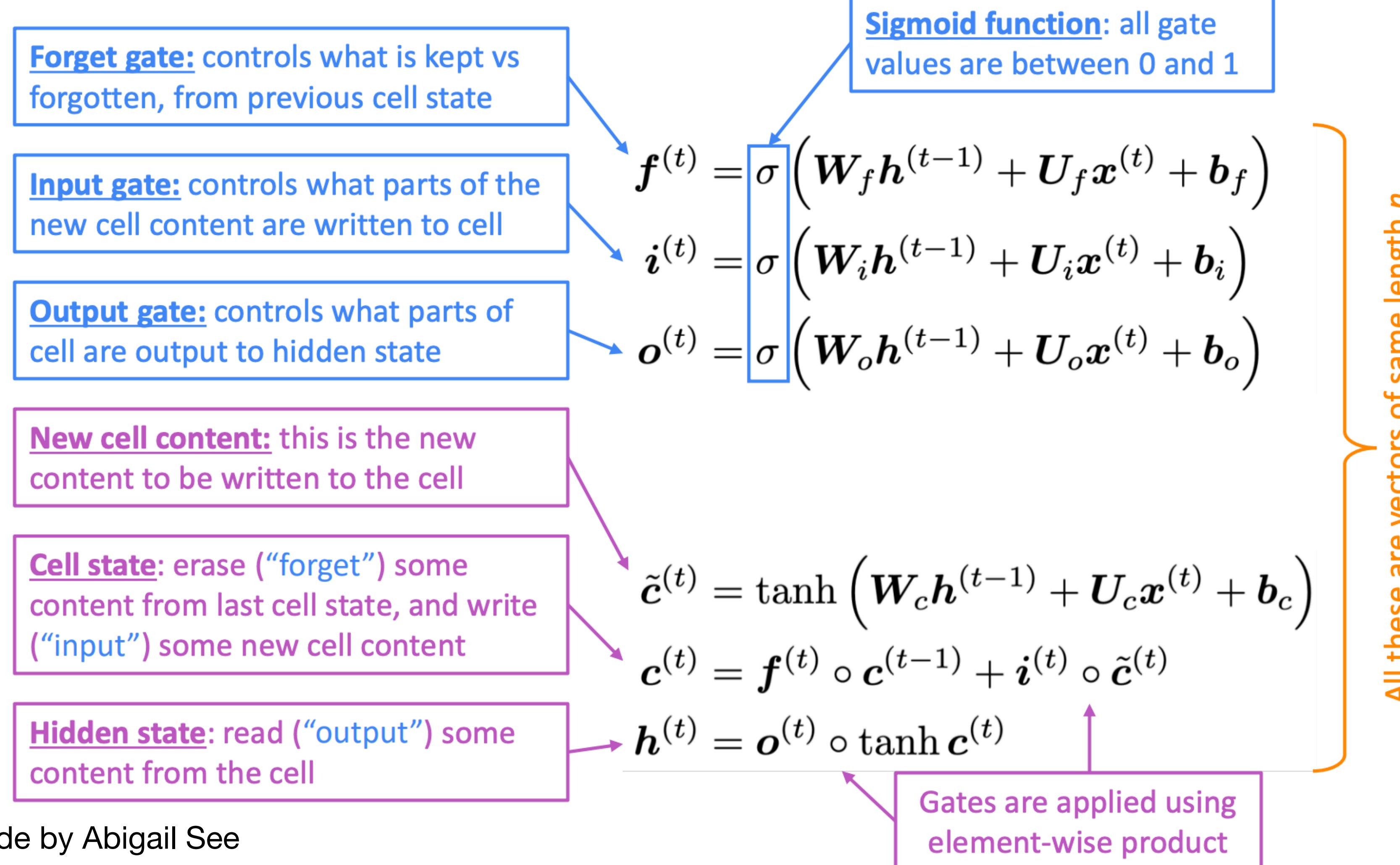
- Introduced by Hochreiter & Schmidhuber 1997
- Separate memory cell  $c$  and hidden state  $h$
- Three gates:
  - **forget gate**: controls what is kept and forgotten from previous cell state
  - **input gate**: controls what part of the new cell content are written to the cell
  - **output gate**: controls what part of the new cell content are written to the hidden state

# LSTM (Long-Short Term Memory)



# LSTM (Long-Short Term Memory)

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  $t$ :



# GRU vs LSTM

- GRU is more efficient to learn (fewer parameters)
- Which is better?
  - No conclusive evidence that one is always superior to the other
- LSTM is typically a good starting choice
- Suggestion: switch to GRU if you want a more efficient model

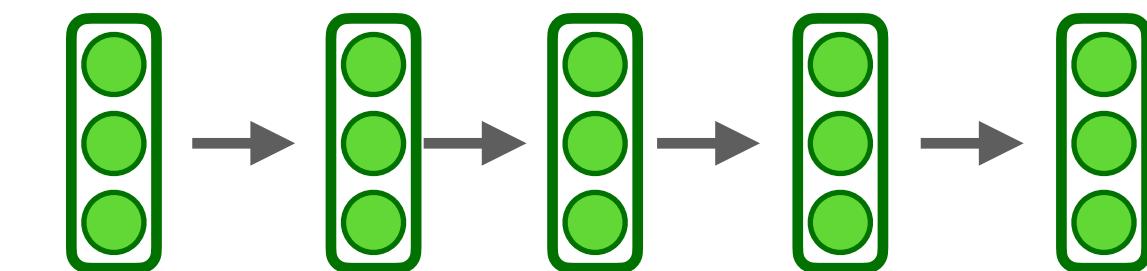
# Interim summary

- ▶ RNNs:
  - ▶ Two fancy variants: **LSTM and GRU** to address the vanishing gradient problem
  - ▶ Other concepts:
    - Bidirectionality, Stacking, Residual connections
  - ▶ Next concepts to cover:
    - ▶ beyond static word embeddings
    - ▶ gluing it all together: attention & contextualised representations



# Today's roadmap

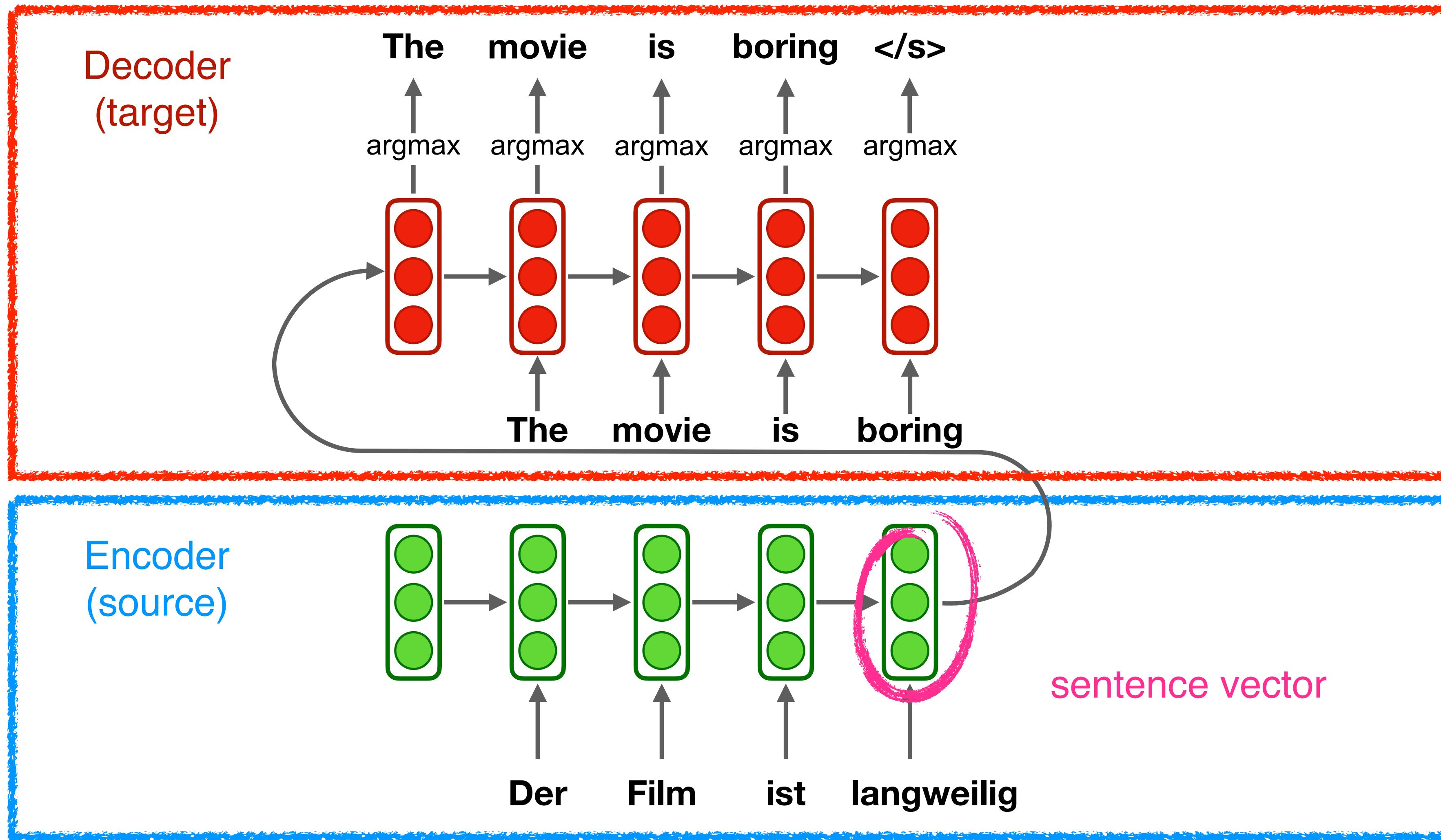
- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



# Attention? Attention!

Many thanks to Lilian Weng for an awesome tutorial (<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>) and  
Graham Neubig's NN for NLP class (<http://www.phontron.com/class/nlp2019/>)

# Motivation: Encoder-decoder model for Machine Translation (MT)

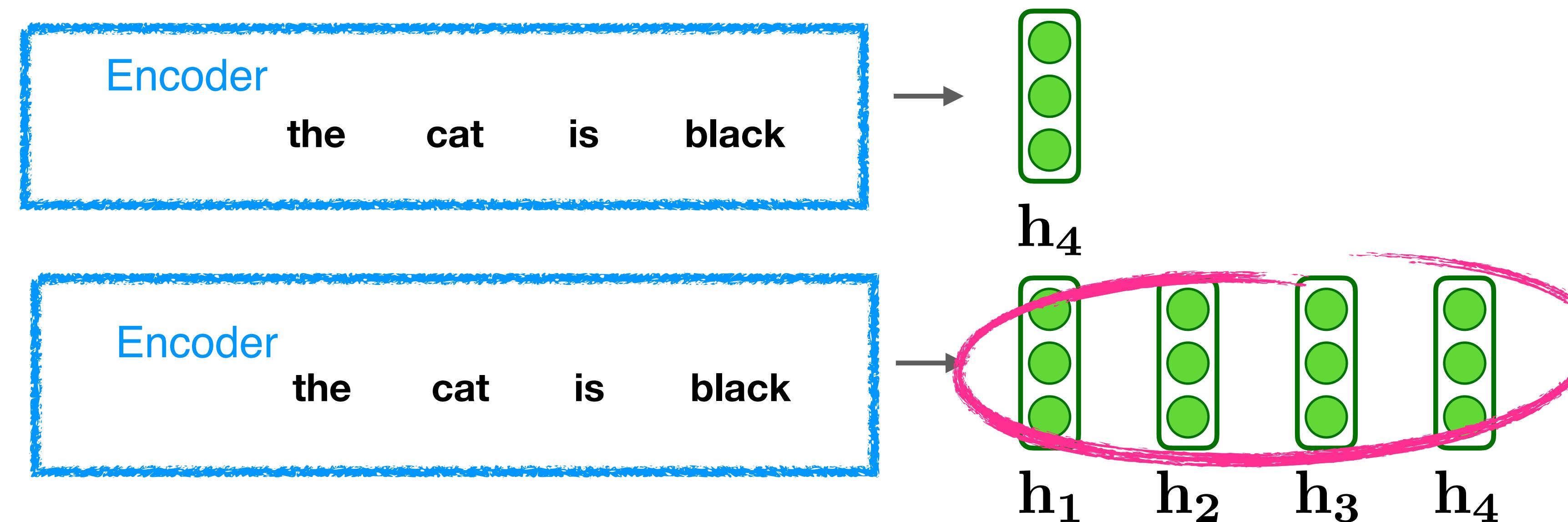


# But: we're cramming it all into..

- The encoder compresses the sentence into a single fixed-size vector. This representation is expected to be a good summary of the entire sentence.
- Disadvantage: incapability of remembering longer sequences.
- **“You can’t cram the meaning of a of a whole %&!\$ing sentence into a single \$&!\*ing vector!”** — Ray Mooney

# Beyond a single static “crammed” vector

- What if we use several vectors, based on the length of the input sequence?
- Idea: when we generate the next word in MT, perhaps we can learn to **attend** to the **relevant** source words



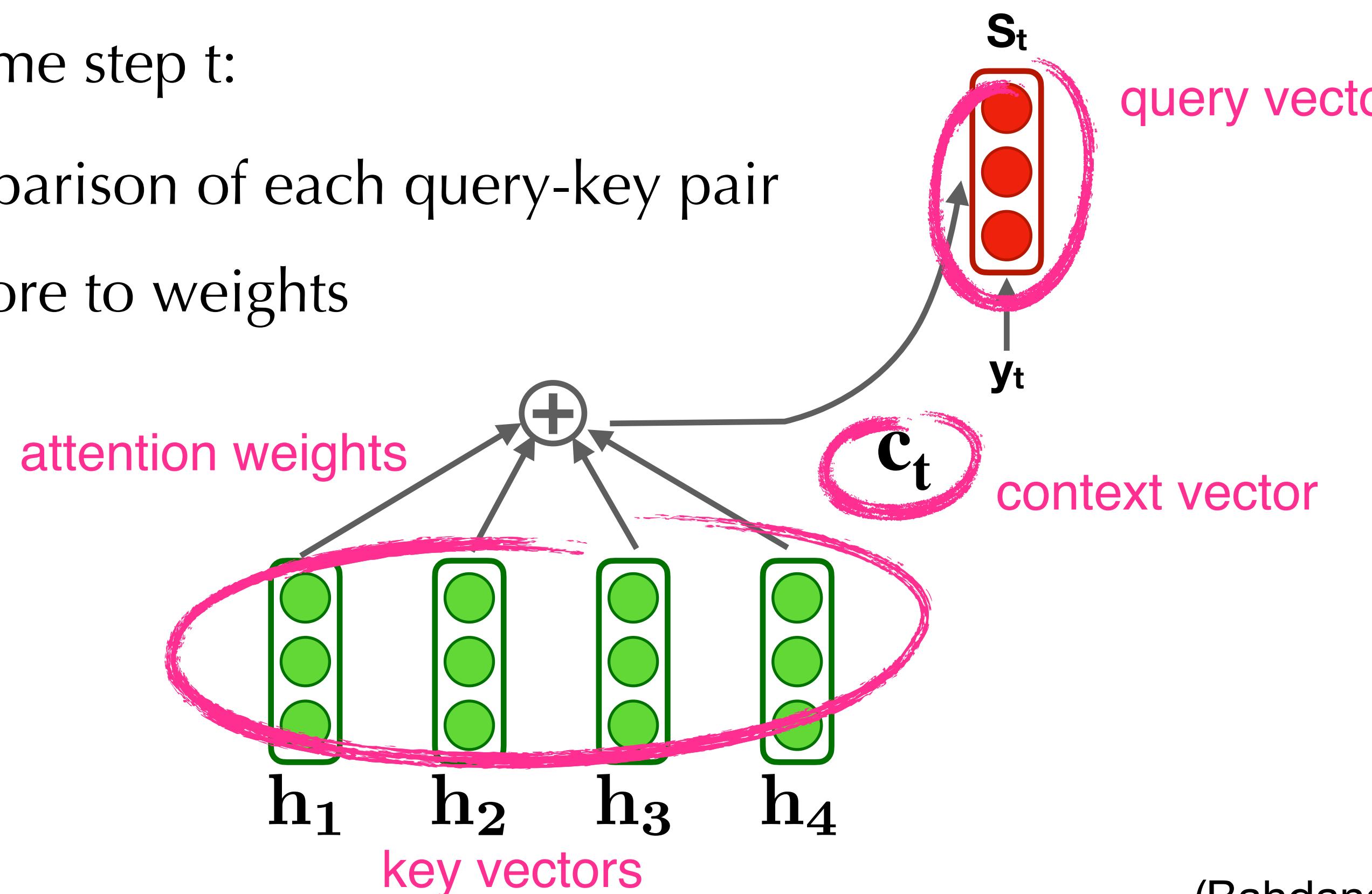
Dynamically look at all encoder hidden states

# Attention: Core Idea

- When decoding, compare query to key vectors and perform a linear combination of the encoded input vectors, weighted by “attention weights”

- Illustration at time step t:

- Pairwise comparison of each query-key pair
- Normalise score to weights
- Get new  $c_t$

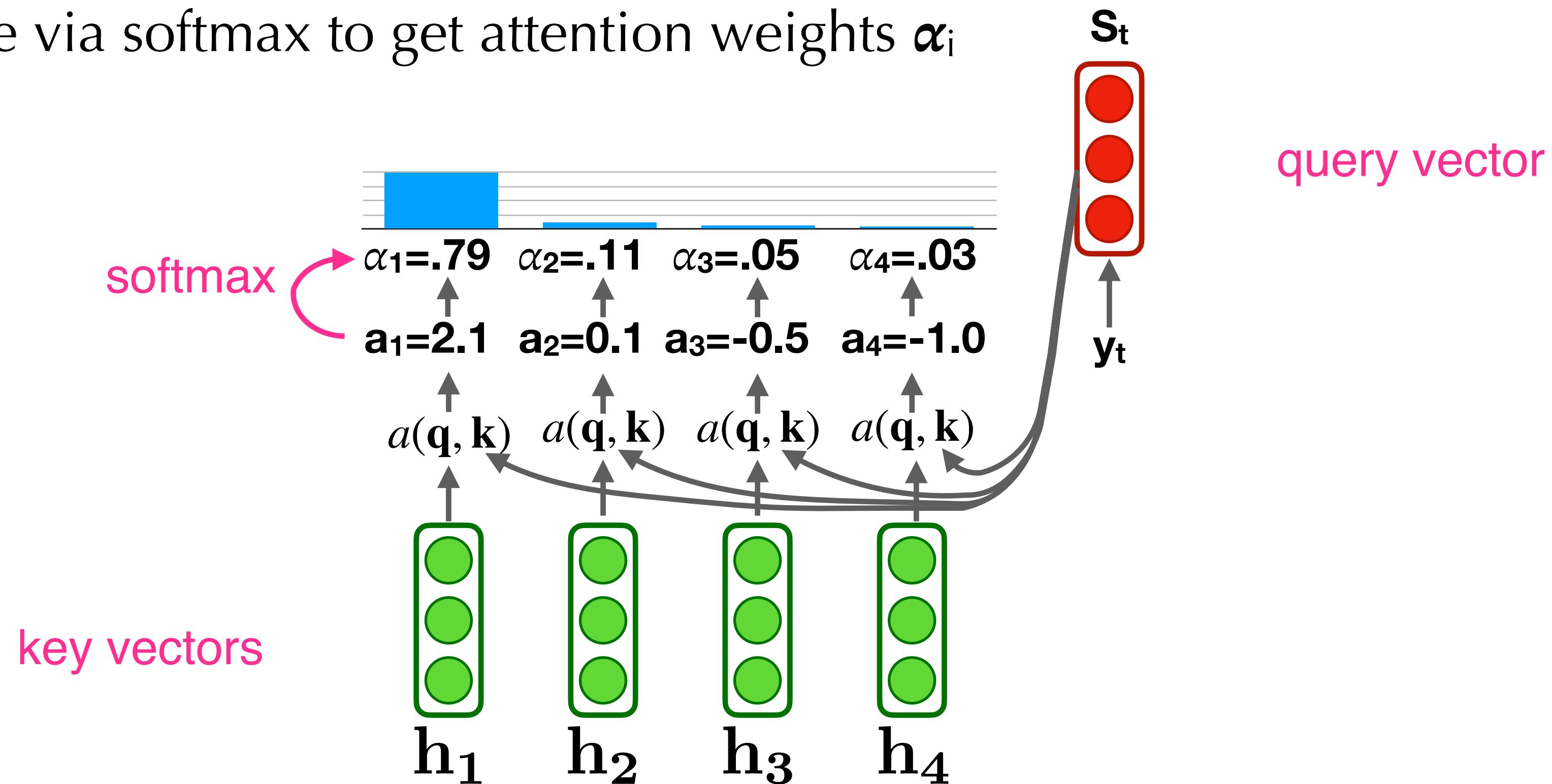


(Bahdanau et al., 2015)



# Calculating attention (1/2): Attention weights $\alpha$

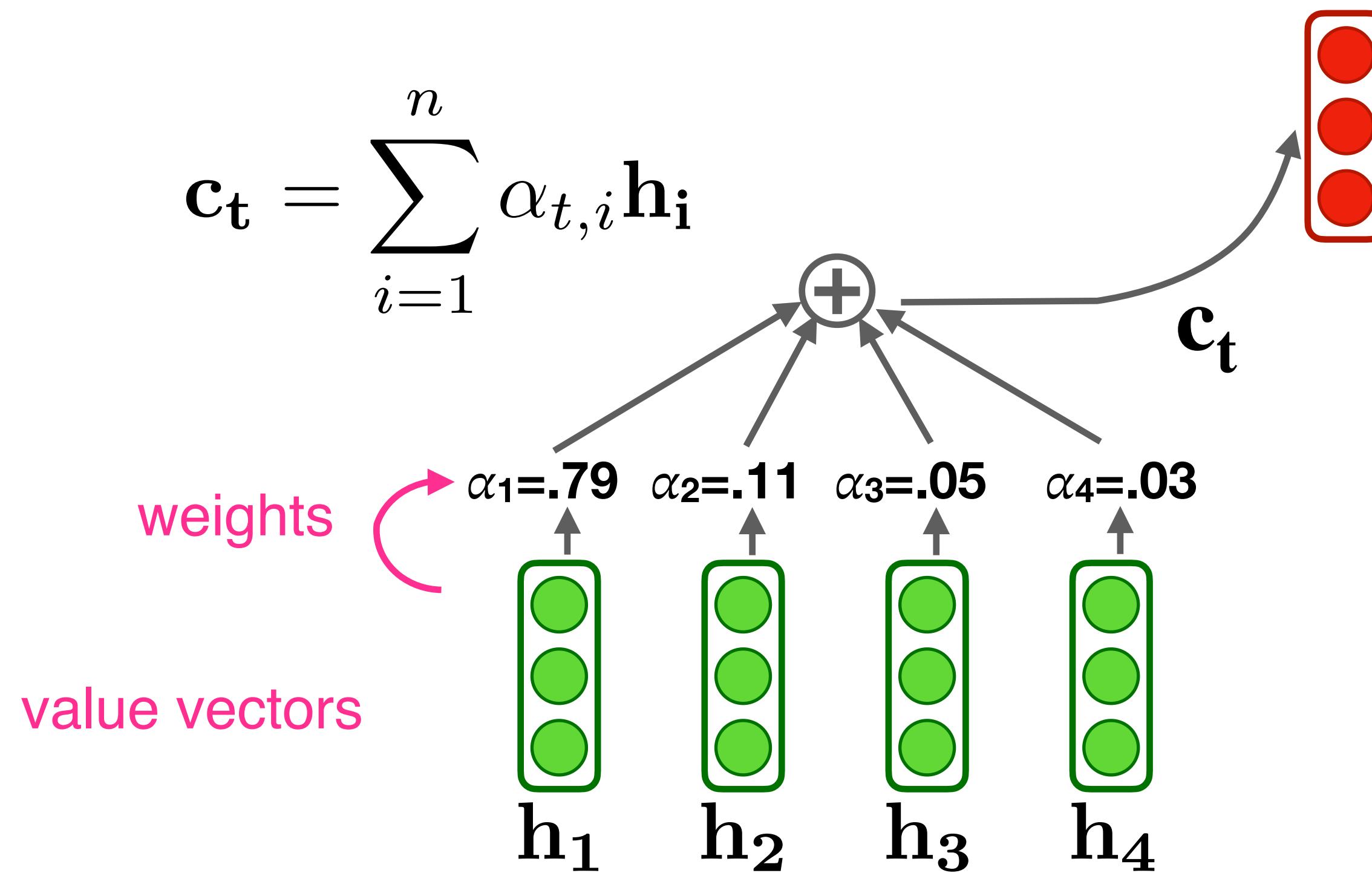
1. For each query-key pair  $(q, k)$ , calculate an **attention score  $a_i$**  by attention function  $a$
2. Normalize via softmax to get attention weights  $\alpha_i$



(Bahdanau et al., 2015)

# Calculating attention (1/2): Attention weights $\alpha$

3. Combine together **value vectors** via attention-weighted sum to get  $\mathbf{c}_t$



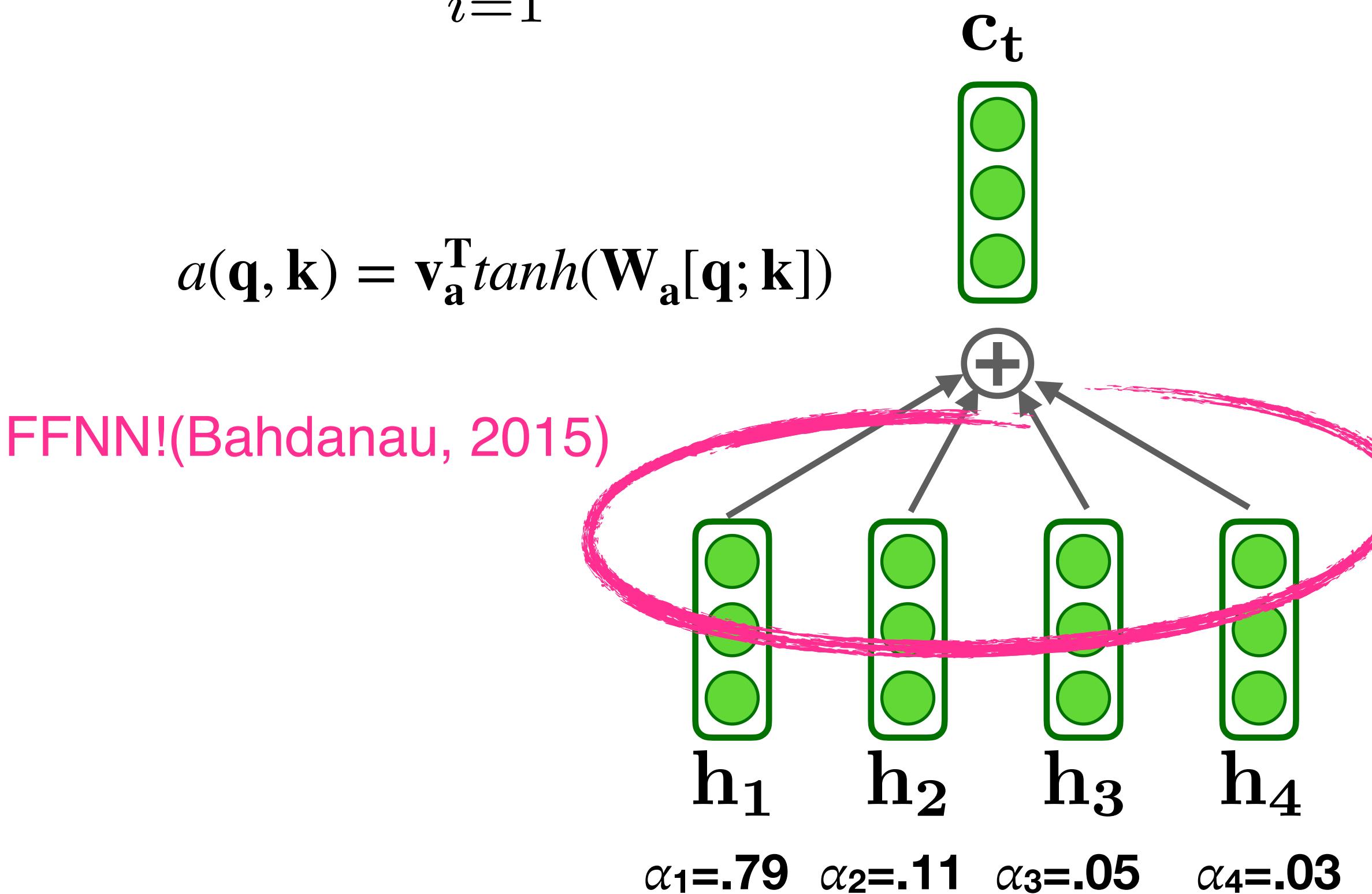
(Bahdanau et al., 2015)

# Attention with a FFNN/MLP

## (Bahdanau, 2015)

$$\mathbf{c}_t = \sum_{i=1}^n \alpha_{t,i} \mathbf{h}_i$$

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{v}_a^T \tanh(\mathbf{W}_a[\mathbf{q}; \mathbf{k}])$$



1. For each query-key pair, calculate score  $a_i$
2. Normalize via softmax to get weights
3. Combine together value vectors via weighted sum to get  $\mathbf{c}_t$
4. Use  $\mathbf{c}_t$  in your model

Different attention functions  $a$   
(e.g., Luong et al., 2015)

# Different attention functions $a()$

- ▶ **Dot product** (Luong et al., 2015)
  - requires same size; but has no parameters

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$$

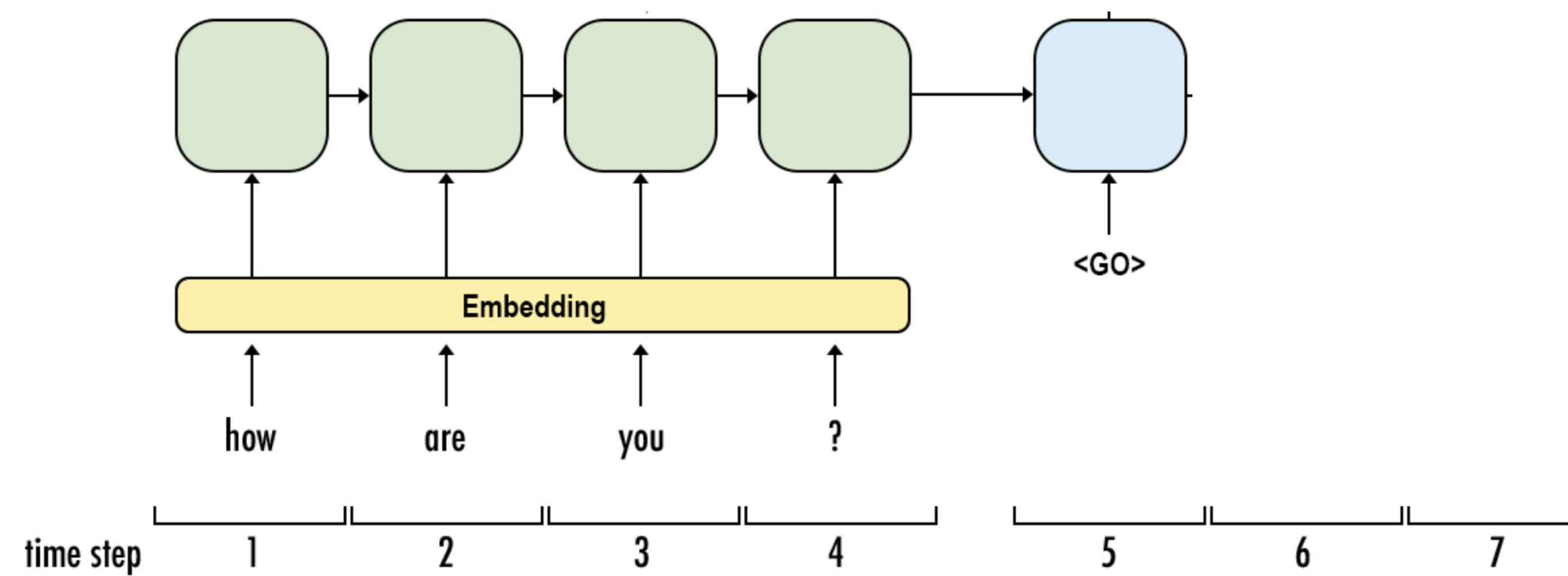
- ▶ **Bilinear** (Luong et al., 2015)

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$$

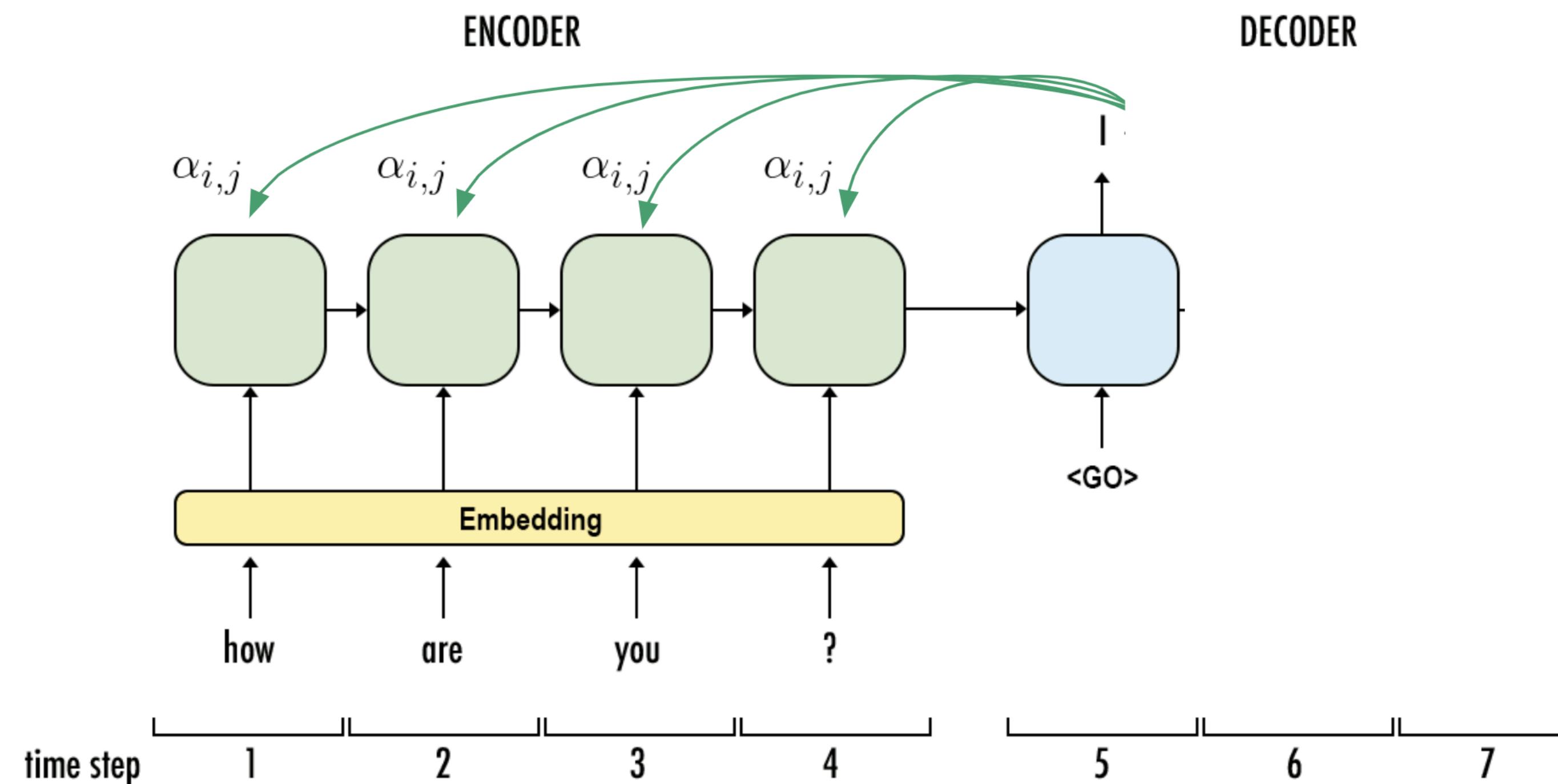
- ▶ **Scaled dot product** (Vaswani et al., 2017)
  - fixes problem of dot product that scale of dot product increases as dimensions get larger

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{n}}$$

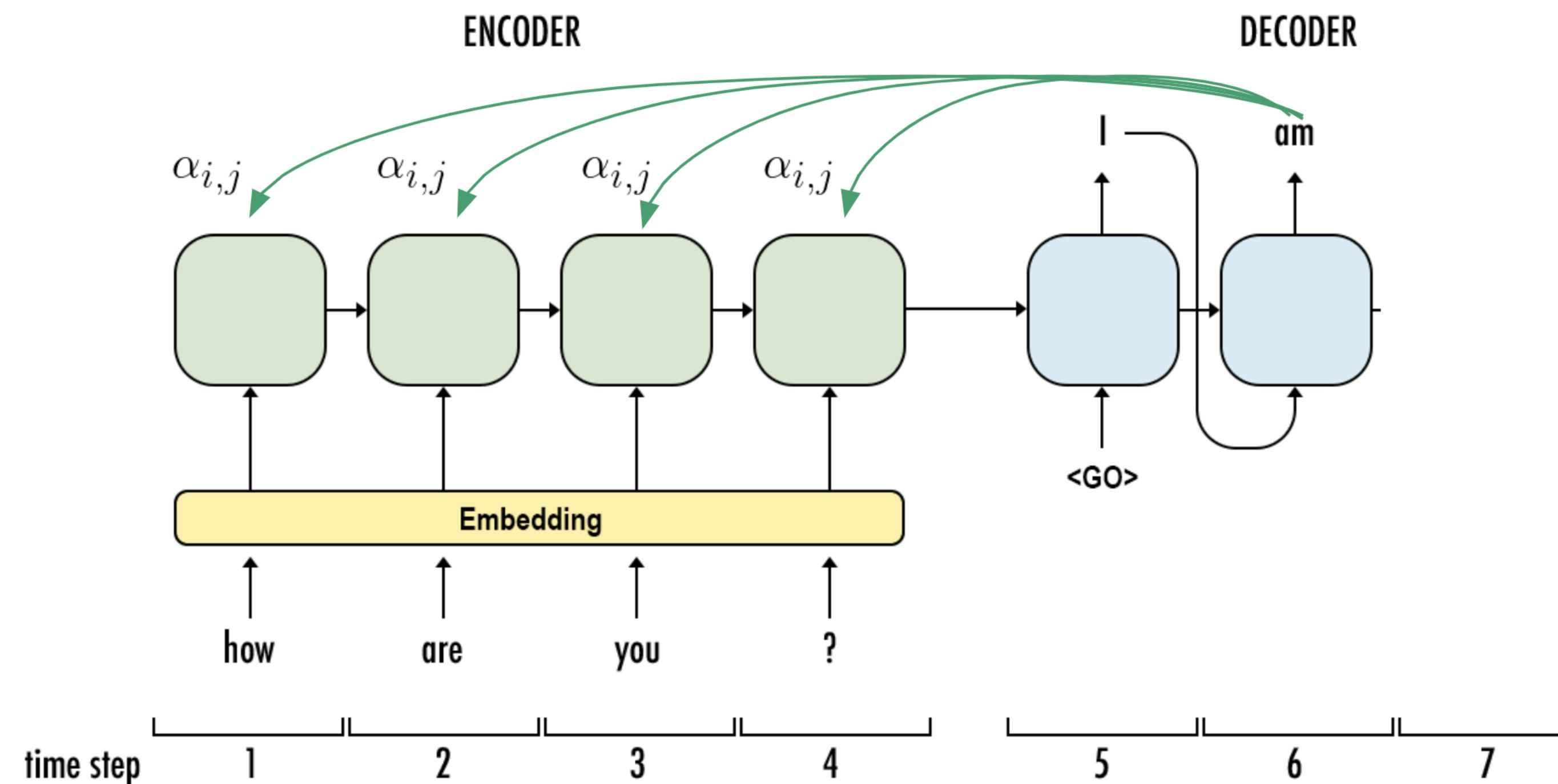
# Recurrent Neural Network



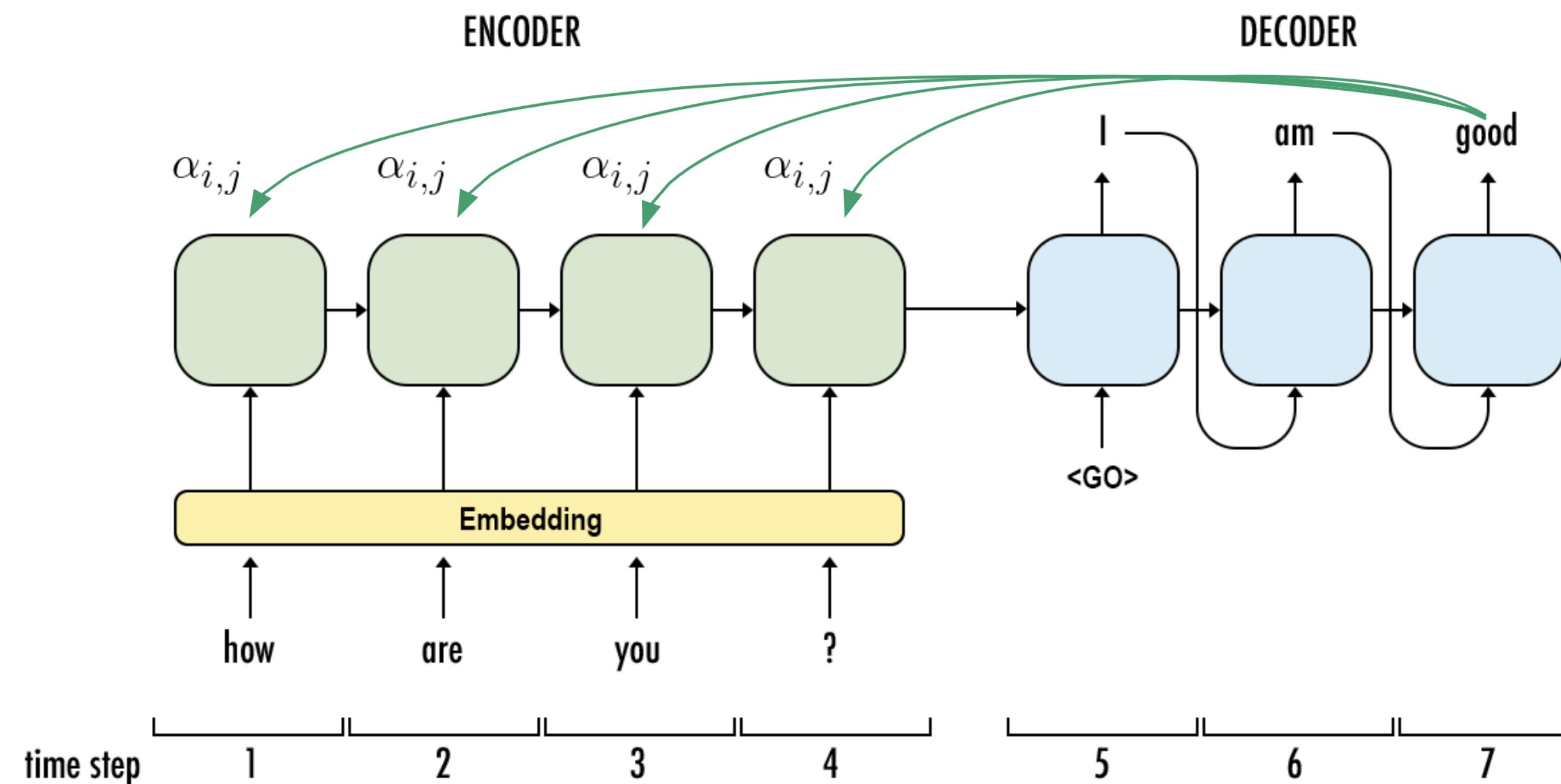
# Recurrent Neural Network With Attention



# Recurrent Neural Network With Attention

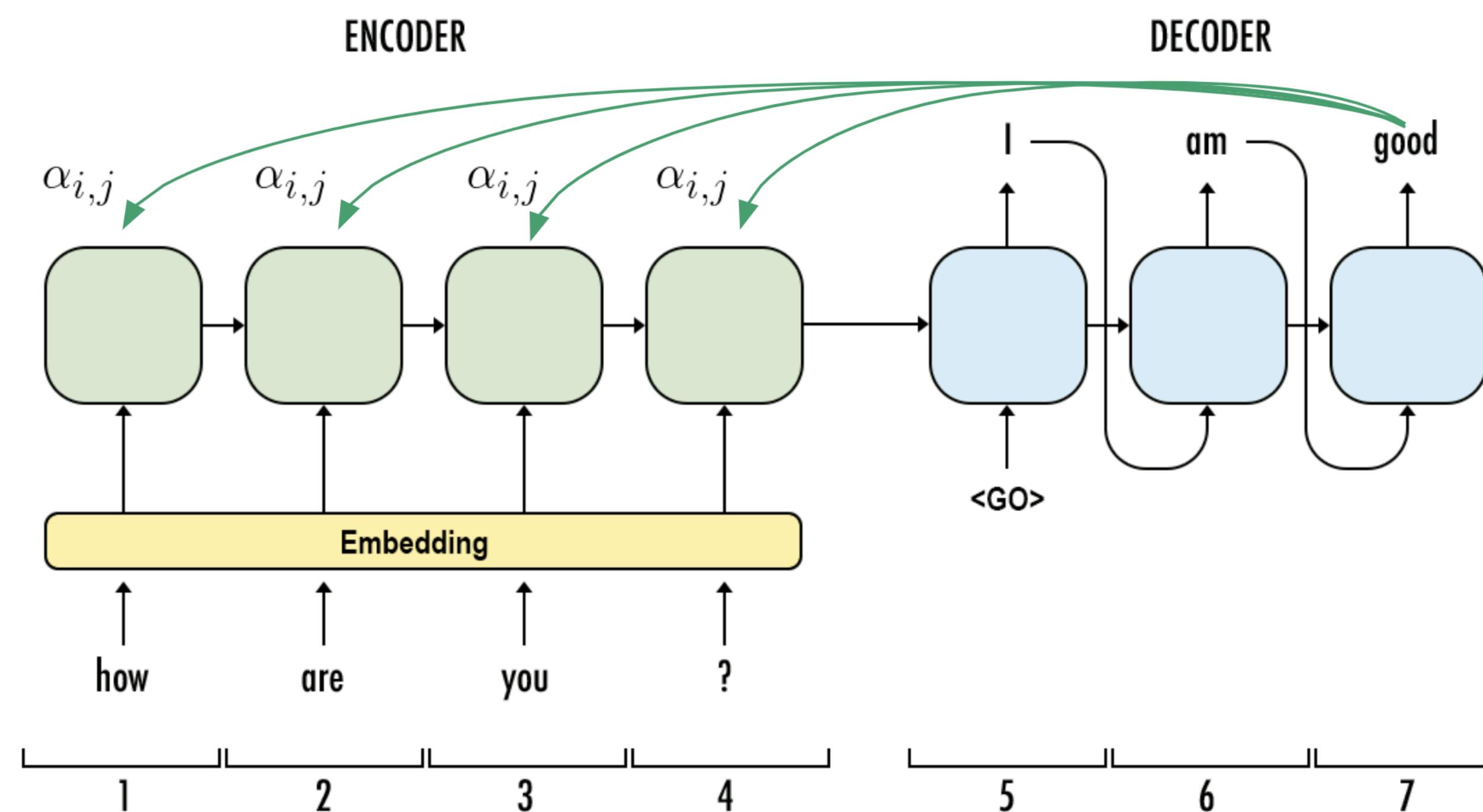


# Recurrent Neural Network With Attention

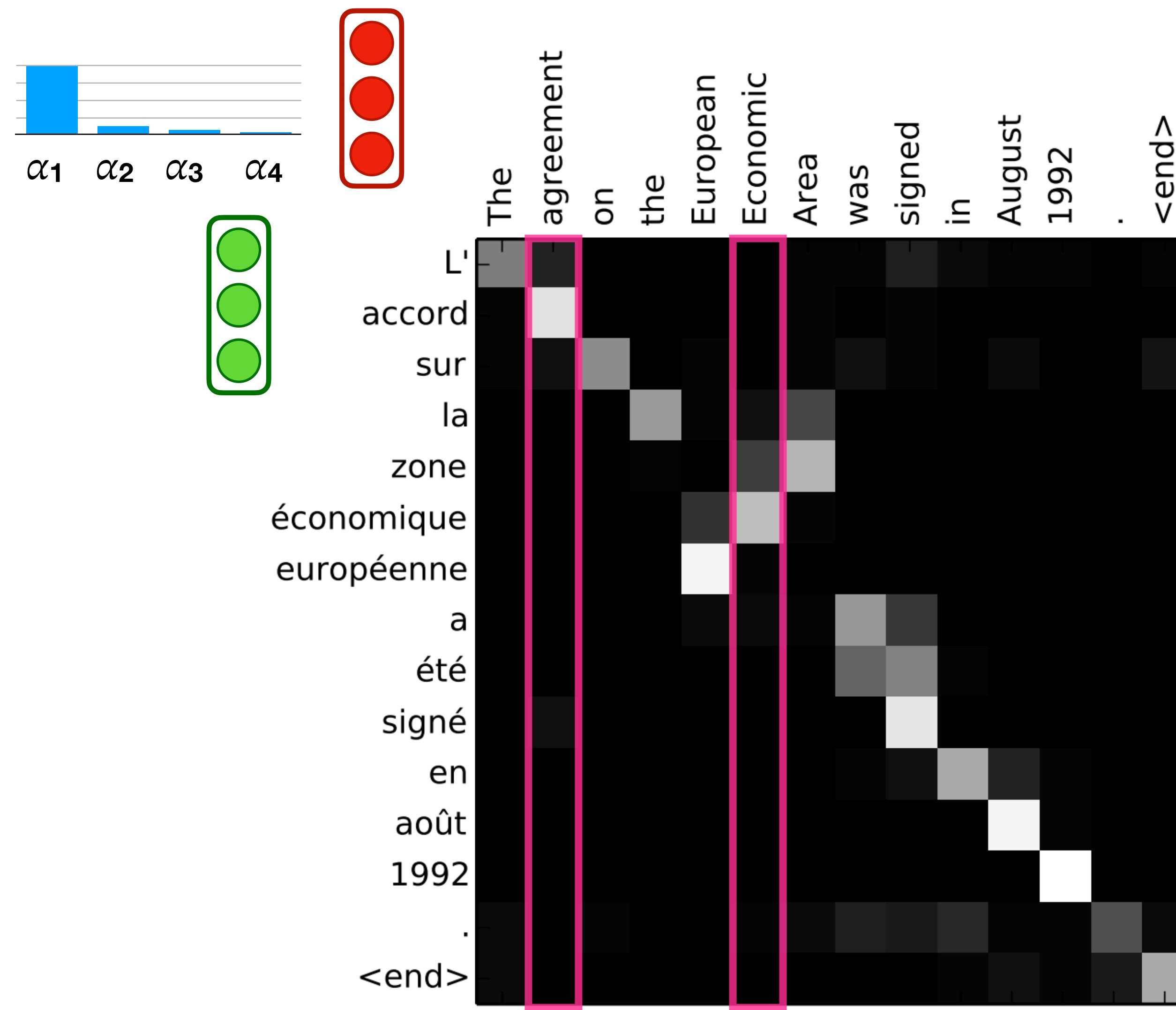


# Recurrent Neural Network With Attention

Encoder-Decoder Attention (aka Cross Attention)

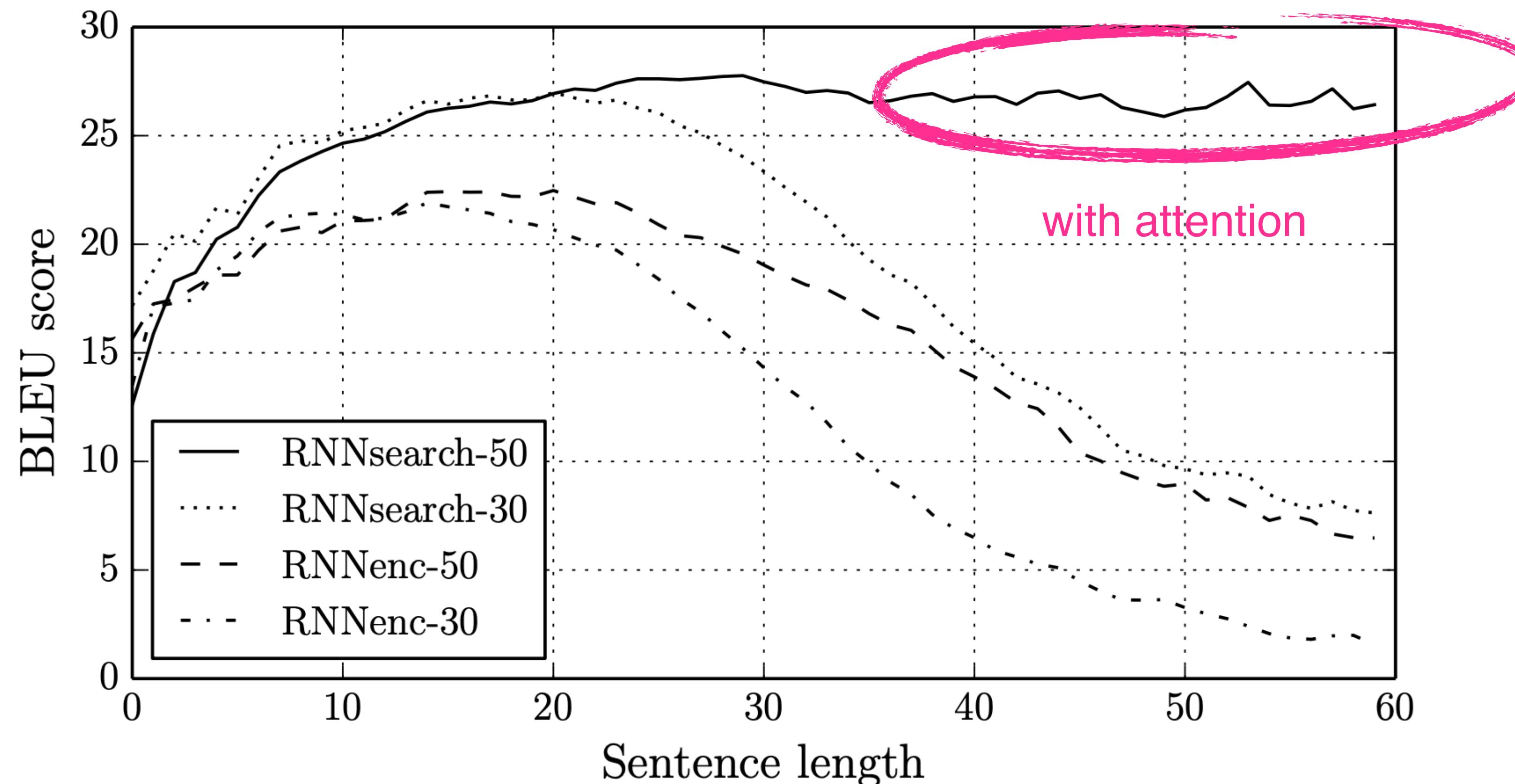


# A graphical example (Bahdanau, 2015)



# Enc-dec performance deteriorates rapidly as input sentence length increases

Cho et al., (2014); Bahdanau et al. (2015)



# Self Attention

- Attend to sentence **itself** (Cheng, Dong, Lapata, 2016) in contrast to cross attention (in encoder-decoder)

## Self Attention

The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .  
The FBI is chasing a criminal on the run .

# Attention in image caption generation

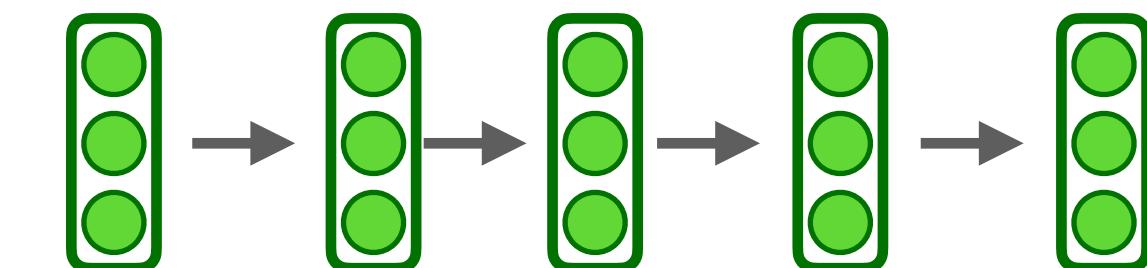
- ▶ Salient parts of the image (e.g., Xu et al., 2015)



A woman is throwing a frisbee in a park.

# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



# Introduction to Contextualized Embeddings (ELMo)

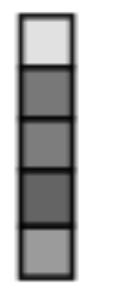
**Welcome to the contextualised embeddings world**

With thanks to Arianna Bisazza, Max Müller-Eberstein & Joris Baan for parts of the slides

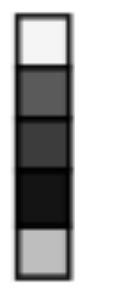
# Traditional (“static”) word embeddings



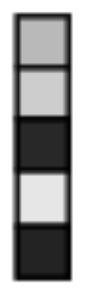
They



ordered



a



Danish



in



Danish



# What is missing?

# No context

I'd like to order a Danish in Danish.

Danish (noun): pastry typical to Denmark

Danish (noun): Scandinavian language spoken in Denmark

# ELMo: Embeddings from Language Models

- A seminal paper to start a new era of representations: **contextualised** embeddings
- **Key Idea:** Learn word token vectors (not type) using entire context
  - Aka token embedding, which depends on surrounding context during use
- ELMo: Self-supervised, pre-trained, RNN-based

Our representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence.

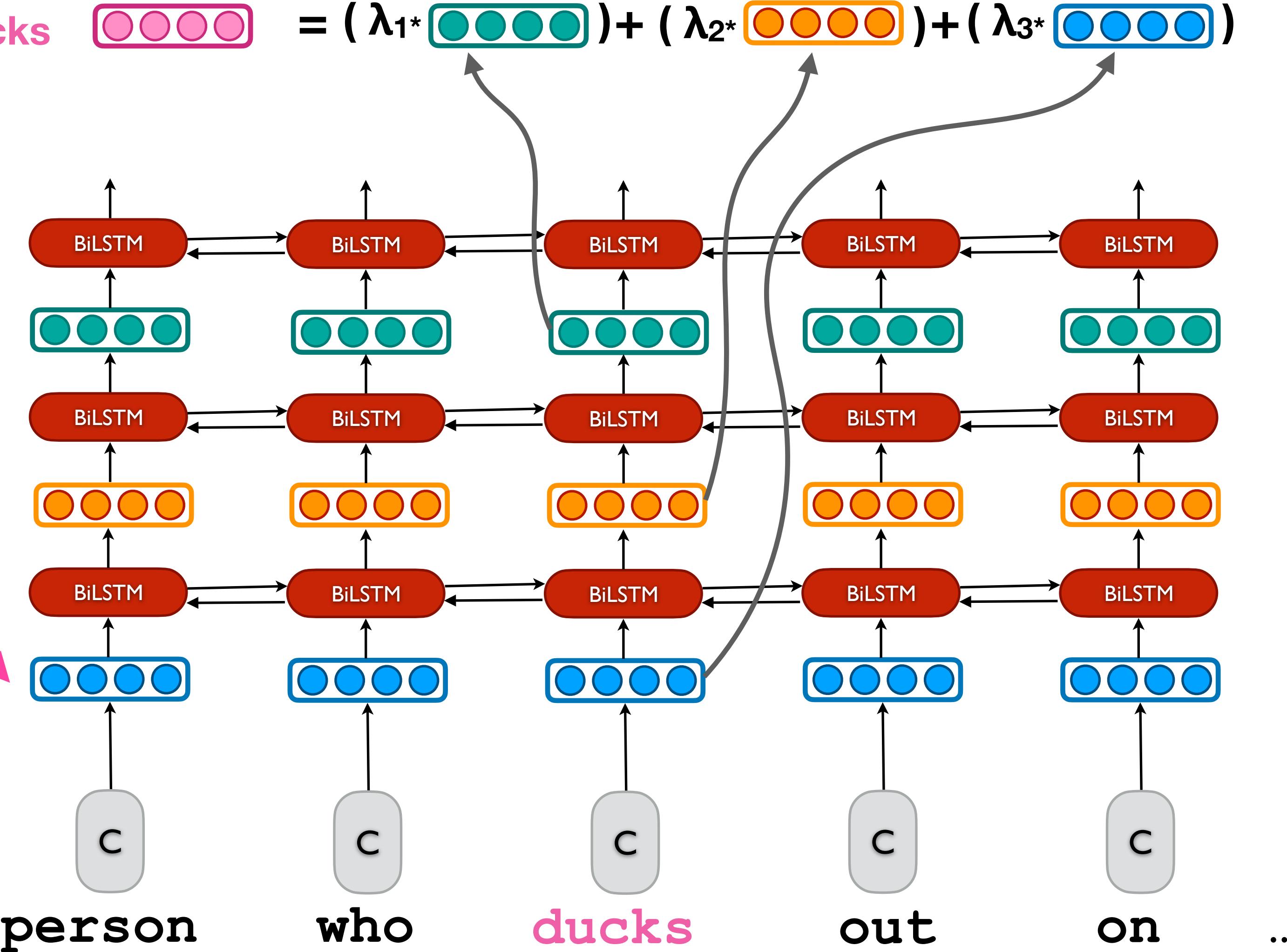
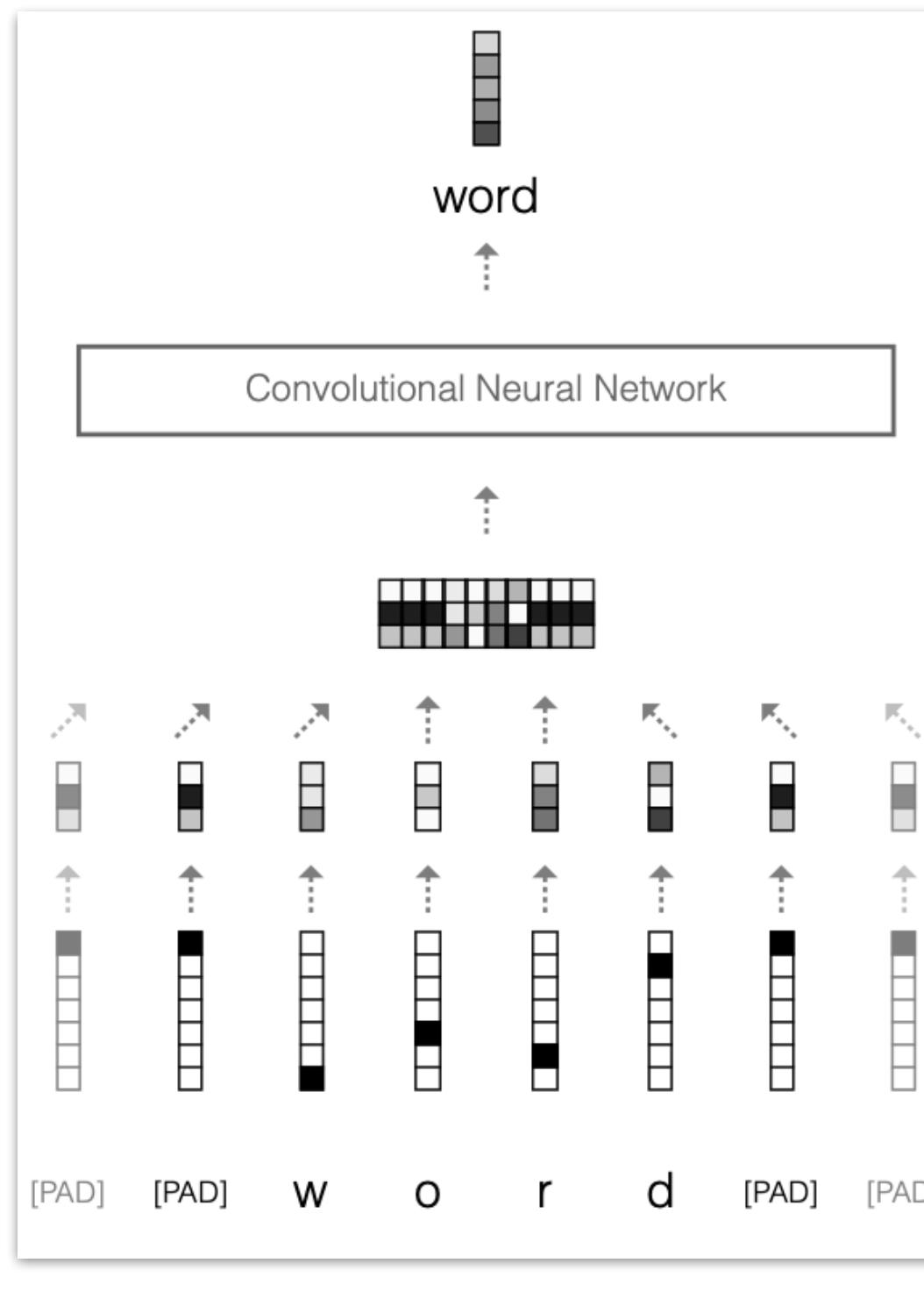
[...] ELMo representations are deep [...]

– Peters et al. (2018)

# From characters to token embeddings

# Embeddings from Language Models

**ELMo<sub>ducks</sub>**  = (  $\lambda_1^*$   ) + (  $\lambda_2^*$   ) + (  $\lambda_3^*$   )



# ELMo - Details

- ELMo: every token is assigned a representation that is a function of the entire input sentence ( $L = \# \text{stacked layers}$ )

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \dots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \dots, L\}, \end{aligned}$$

- This gives  $2L+1$  representations - Which to use?
  - Just the top layer (similar to TagLM; Peters et al., 2017)
  - Include all  $L+1$  layers, average
  - All layers, weighted average (best)

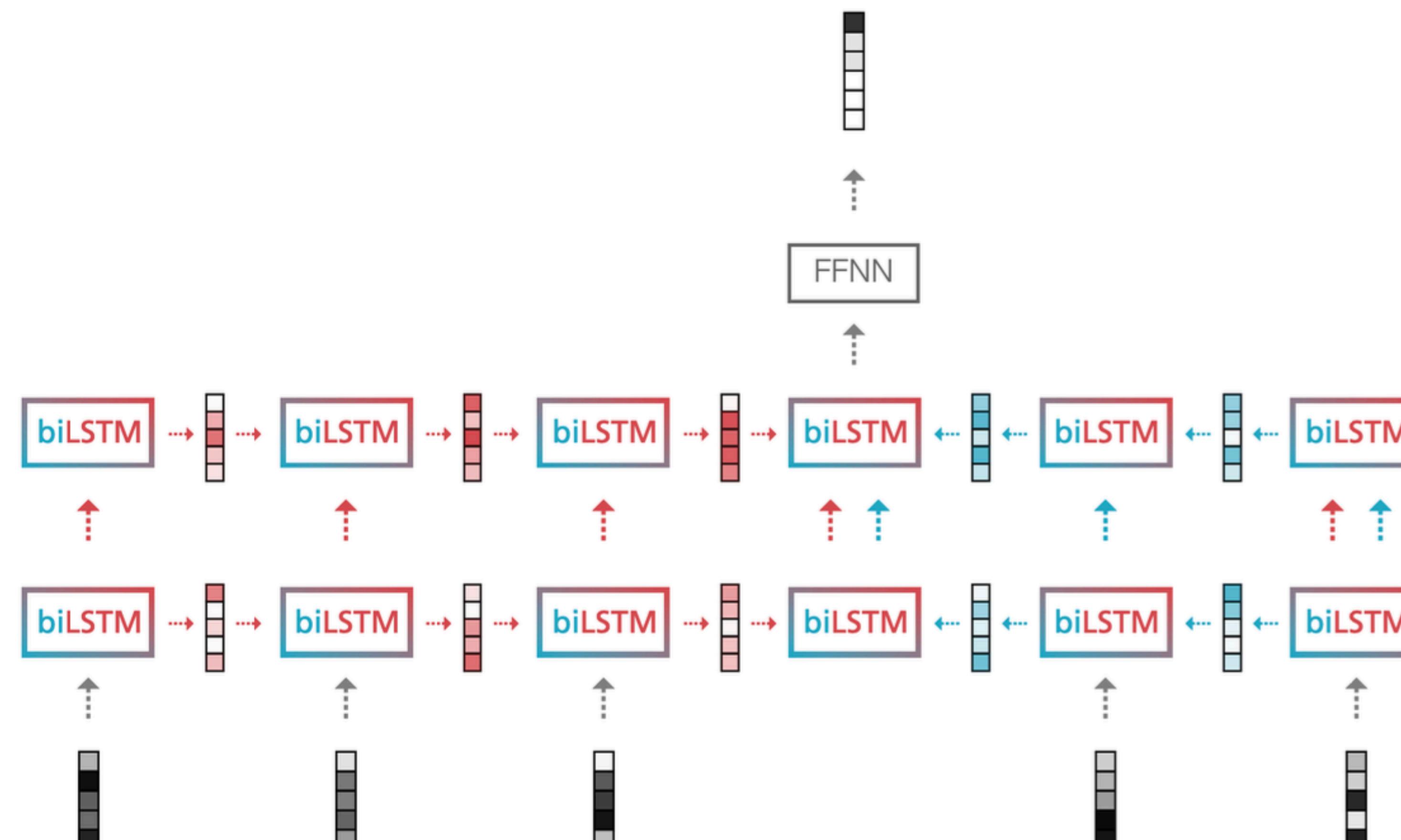
$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^L s_j^{task} \mathbf{h}_{k,j}^{LM}$$

# Training ELMo

# Language Modeling Objective

- Word2vec trains word by word
  - Uses context during training
  - No context during use
- ELMo trains on word sequences
  - Predict each token using its context
  - Softmax layer applied to the top layer's output
  - Uses sequences as input during use

Danish



They

ordered

a

in

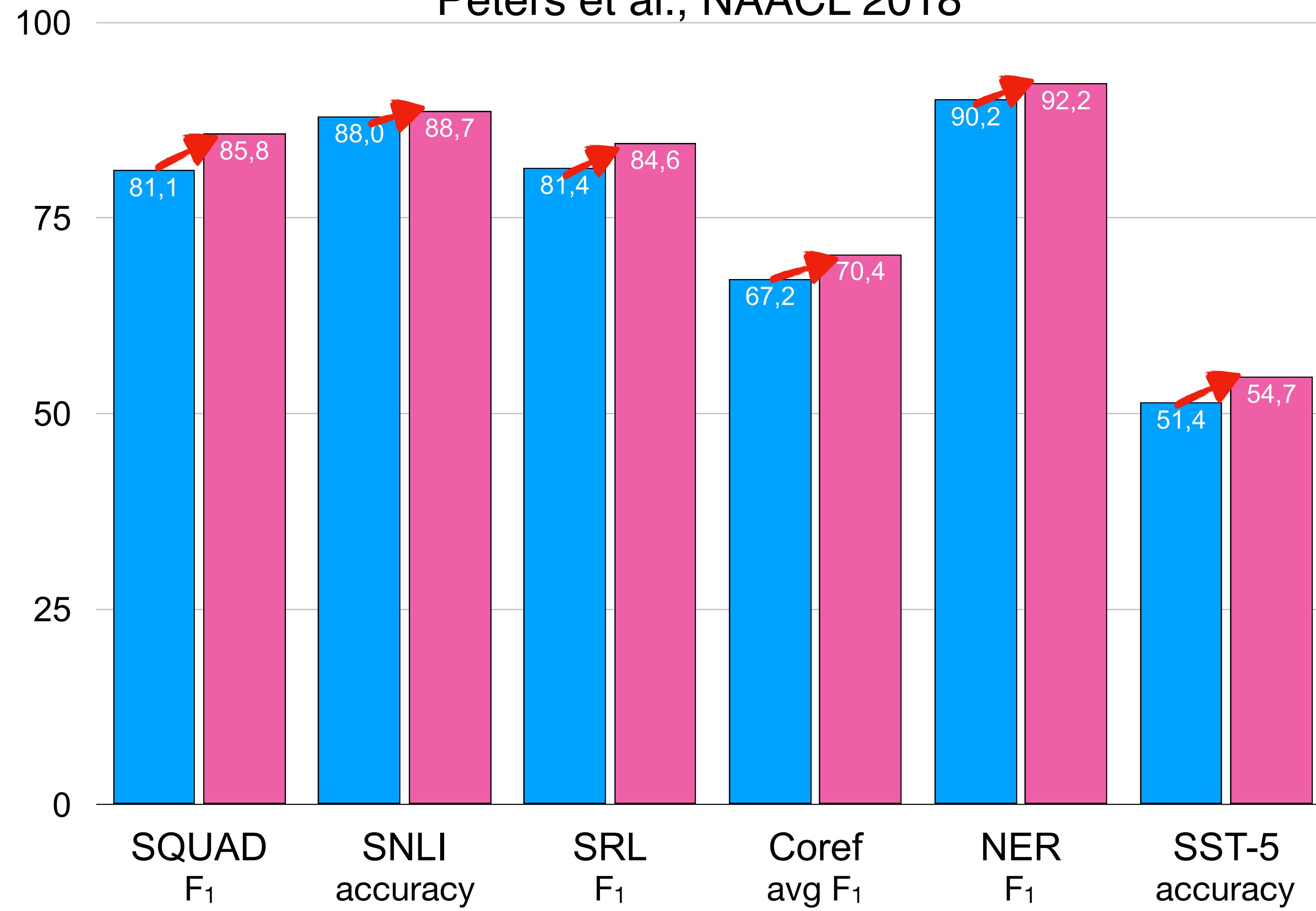
Danish

# How to use ELMo for your task?

- ▶ Recipe: For a given instance
  - ▶ Run biLM to get the representations for each word
  - ▶ Concatenate ELMo embeddings into task-specific model, e.g.,
    - ▶ as additional input to static word embeddings
    - ▶ as additional hidden representation
    - ▶ ... many choices, best might depend on end task

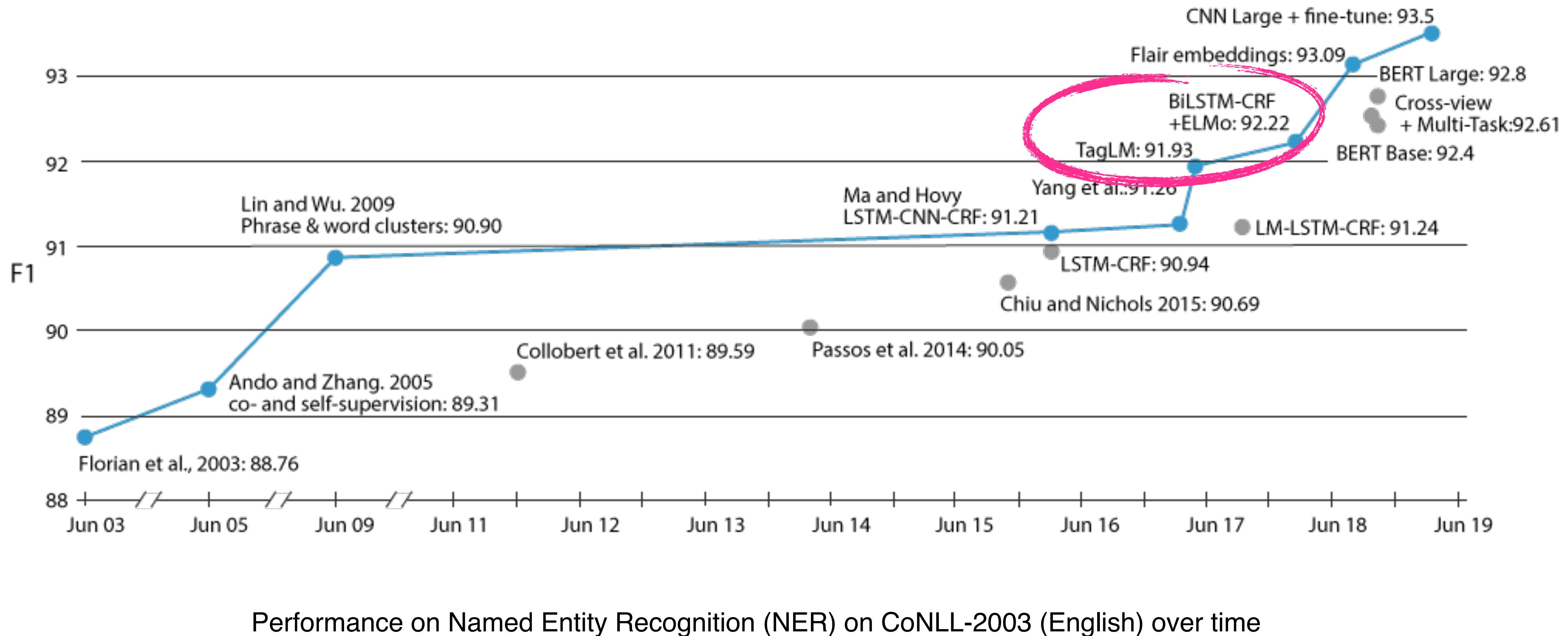
# Results over 6 NLP benchmarks

Peters et al., NAACL 2018



# NLP Progress on NER

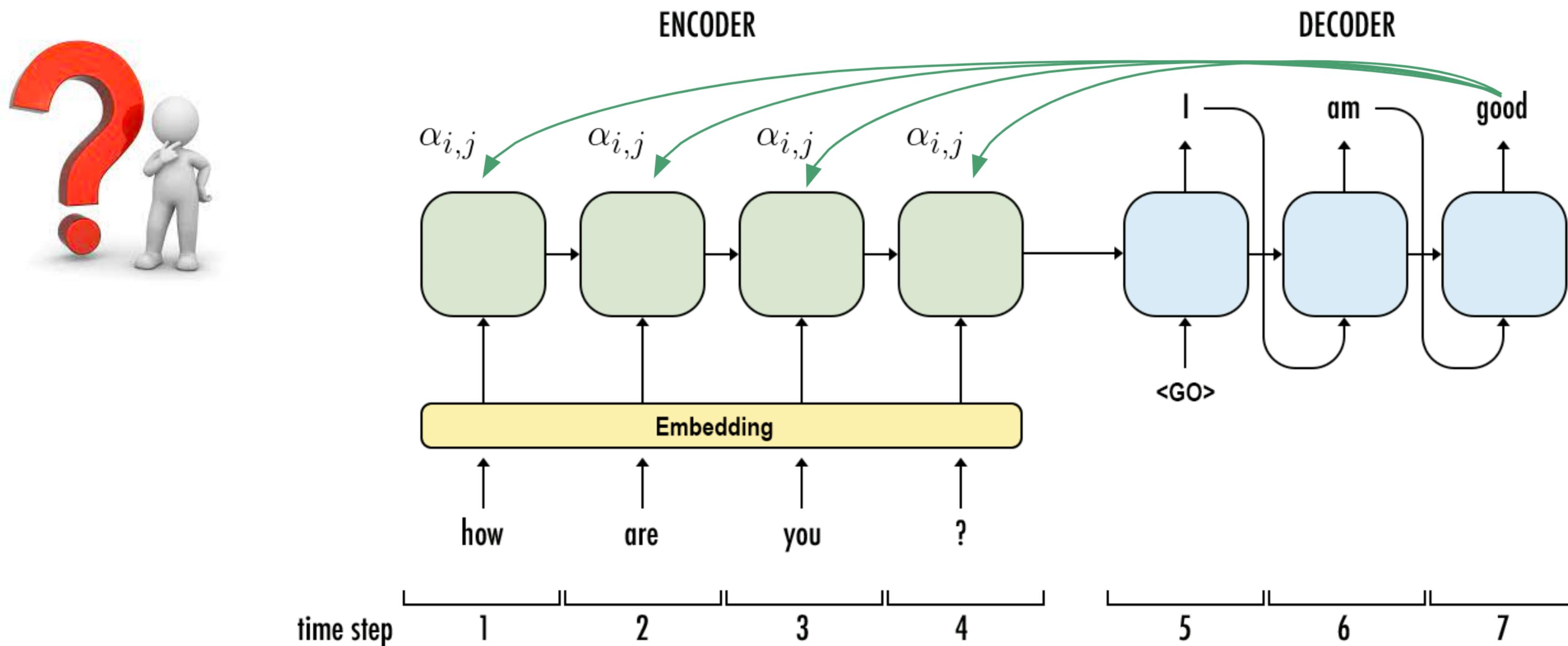
- From Ruder et al.'s 2019 NAACL tutorial



# Is ELMo the first such model? No!

- ELMo is *deeper* compared to an earlier model by Peters et al., 2017 ACL (**TagLM**)
- It doesn't require parallel data (as an earlier model like **CoVe** does, by McCann et al., 2017 NeurIPS)
- ELMo: state-of-the-art performance on 6 benchmarks in 2018

# Limits of RNNs?

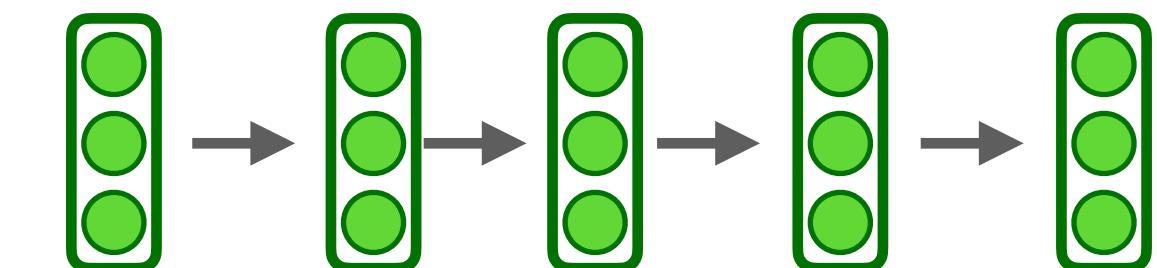


# Transformer & LLMs

Part III

# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ **The Transformer**, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



# Transformers

A Revolution in NLP and ML



# What's next

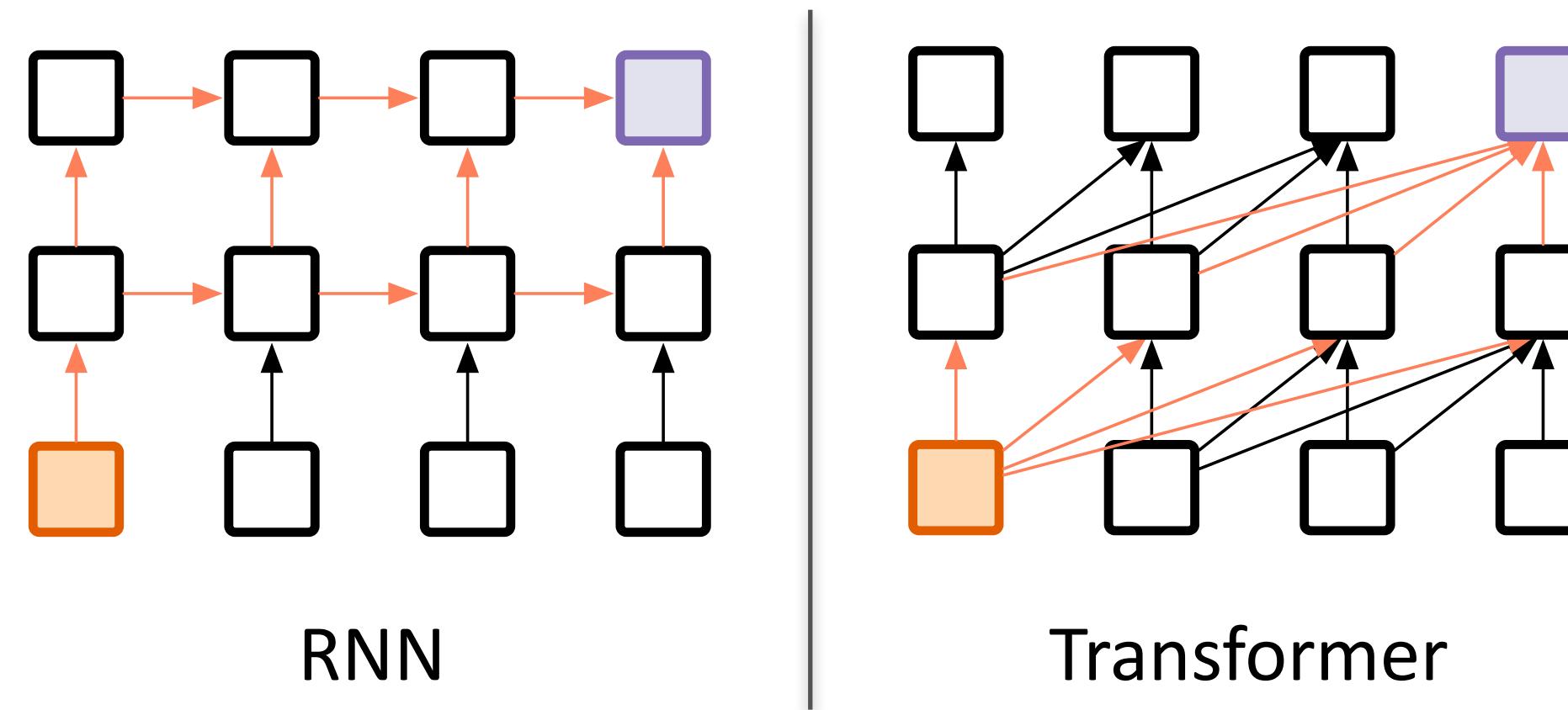
- The Transformer
  - High-level Architecture
    1. Encoder
    2. Decoder
    3. Multi-Head (Self) Attention



# FULLY ATTENTIONAL NETWORKS (A.K.A. TRANSFORMER)

# Core idea: Attention is All You Need (Vaswani et al. 2017)

- Attention has major impact on seq2seq performance
  - Recurrency is an obstacle to parallelization
- => Can we build a fully attentional seq2seq model without recurrency?



Slides by Arianna Bisazza



# The Transformer

## Machine Translation

Attention Is All You Need

- “**Transforms**” one sequence into another
- No convolutions, no recurrence (no time steps), only **attention**
- **Highly parallelizable**
- Capture long range dependencies
- Better quality

Ashish Vaswani\*  
Google Brain  
avaswani@google.com

Noam Shazeer\*  
Google Brain  
noam@google.com

Niki Parmar\*  
Google Research  
nikip@google.com

Jakob Uszkoreit\*  
Google Research  
usz@google.com

Llion Jones\*  
Google Research  
llion@google.com

Aidan N. Gomez\* †  
University of Toronto  
aidan@cs.toronto.edu

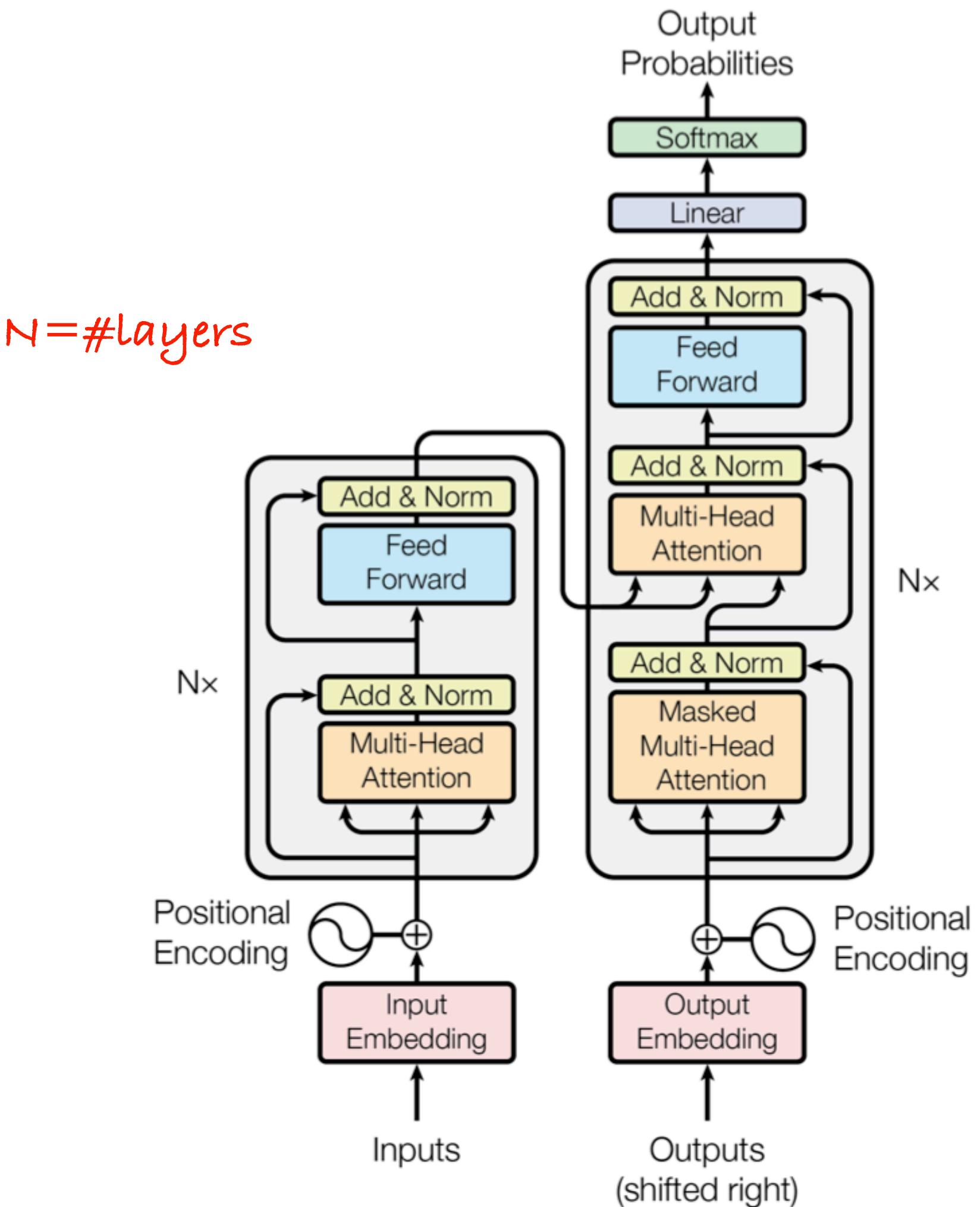
Lukasz Kaiser\*  
Google Brain  
lukaszkaiser@google.com

Illia Polosukhin\* ‡  
illia.polosukhin@gmail.com

### Abstract

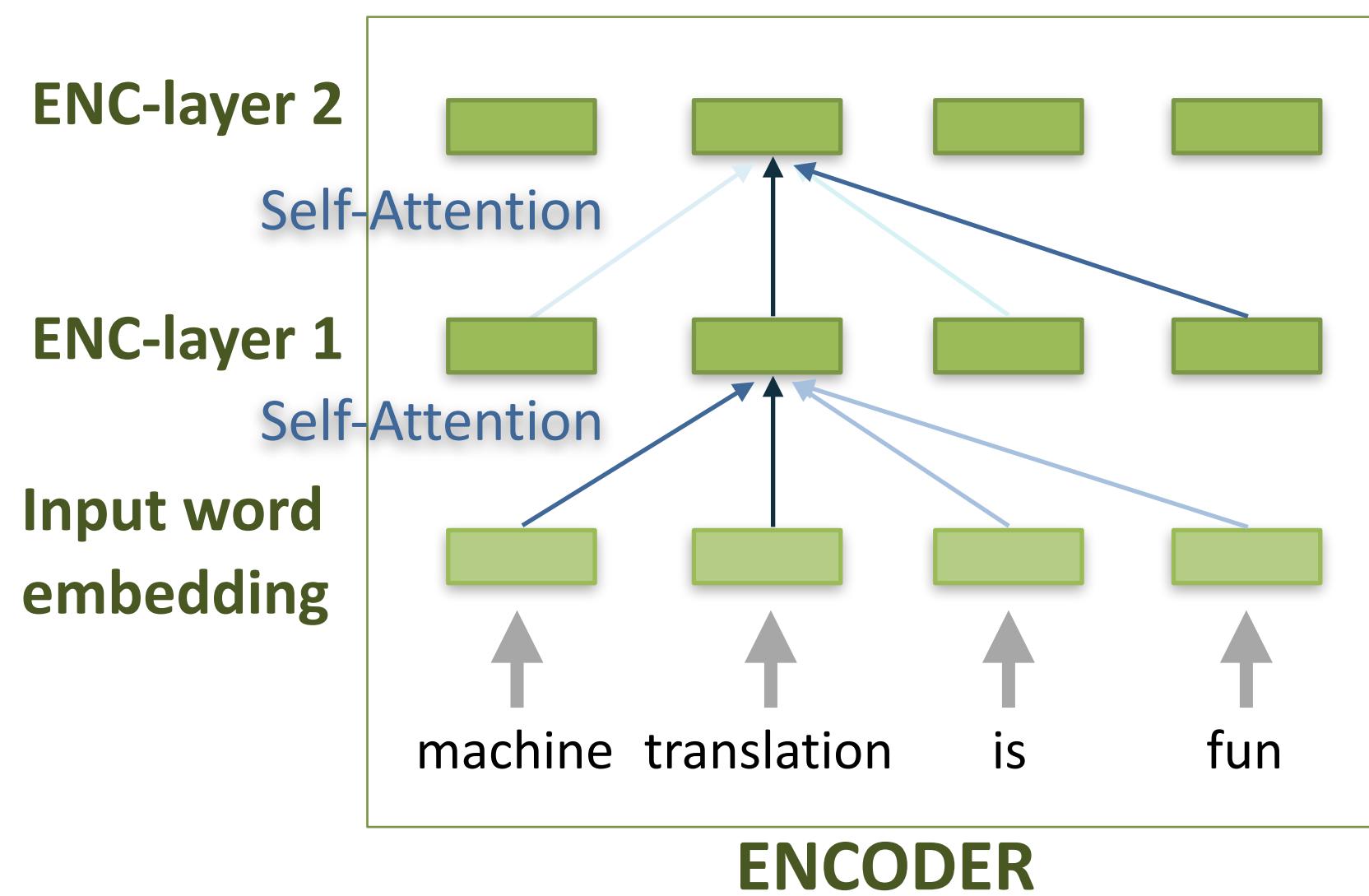
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

# A scary beast

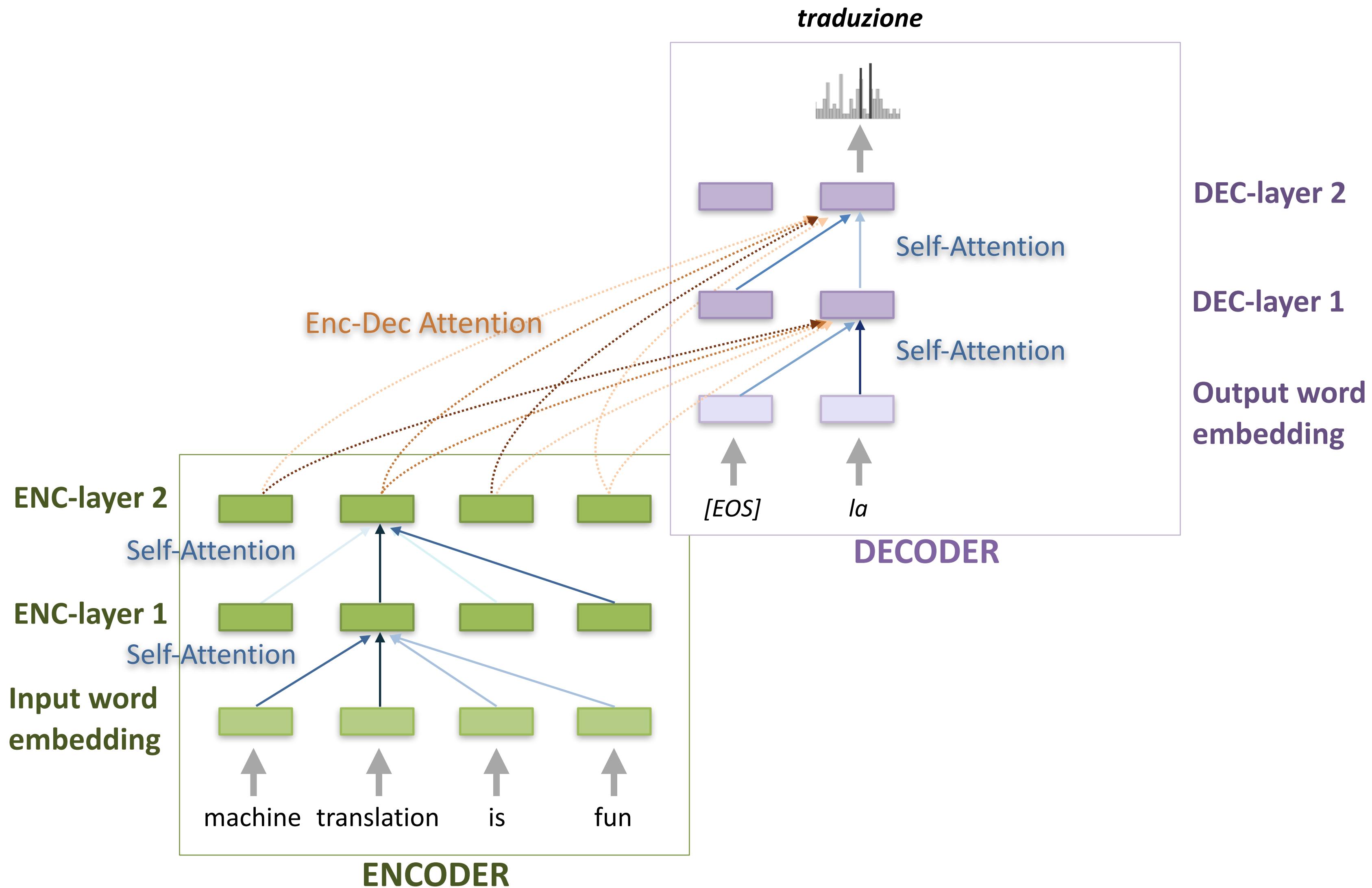


# TRANSFORMER ARCHITECTURE OVERVIEW

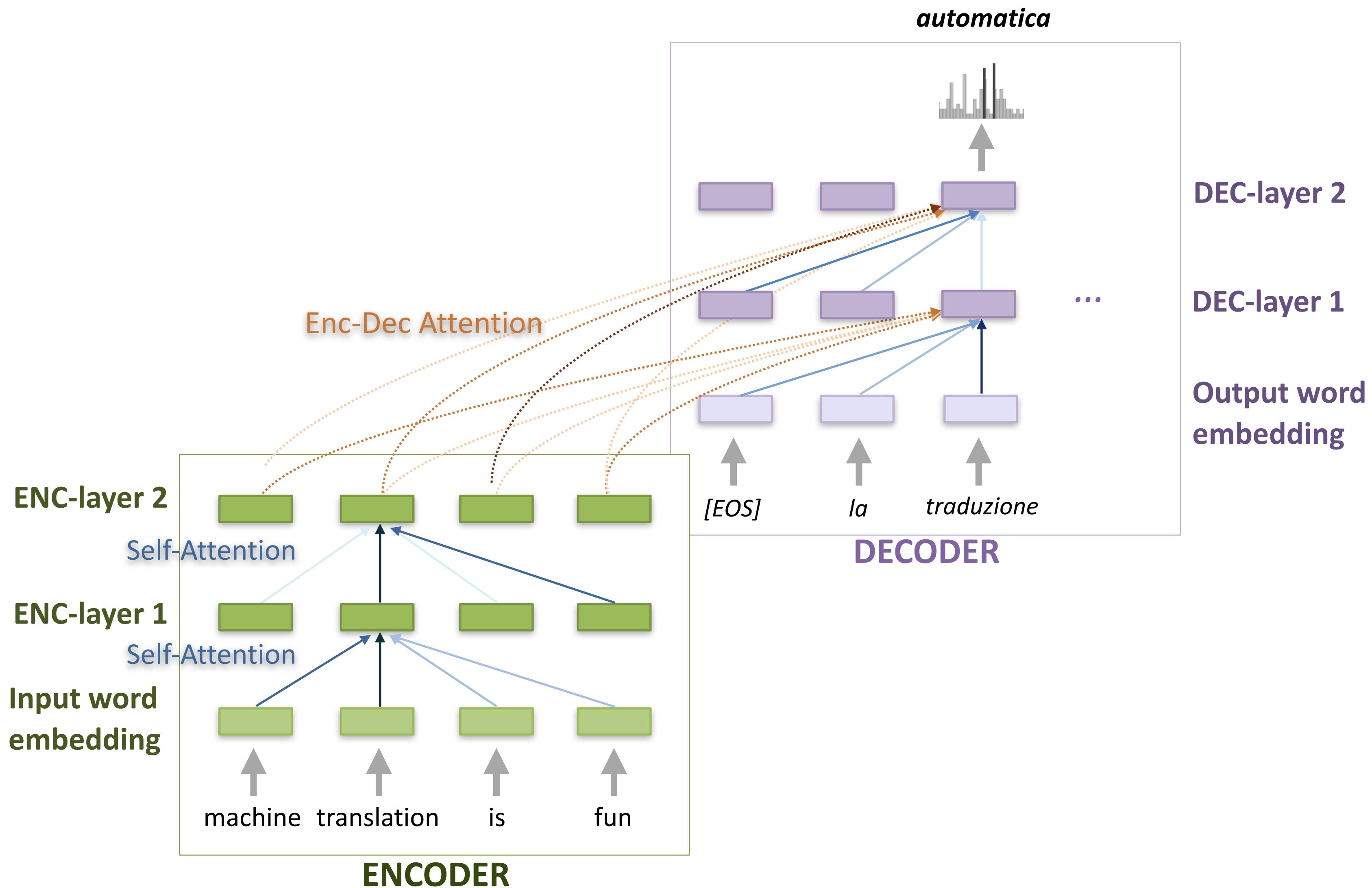
# Transformer Architecture Overview



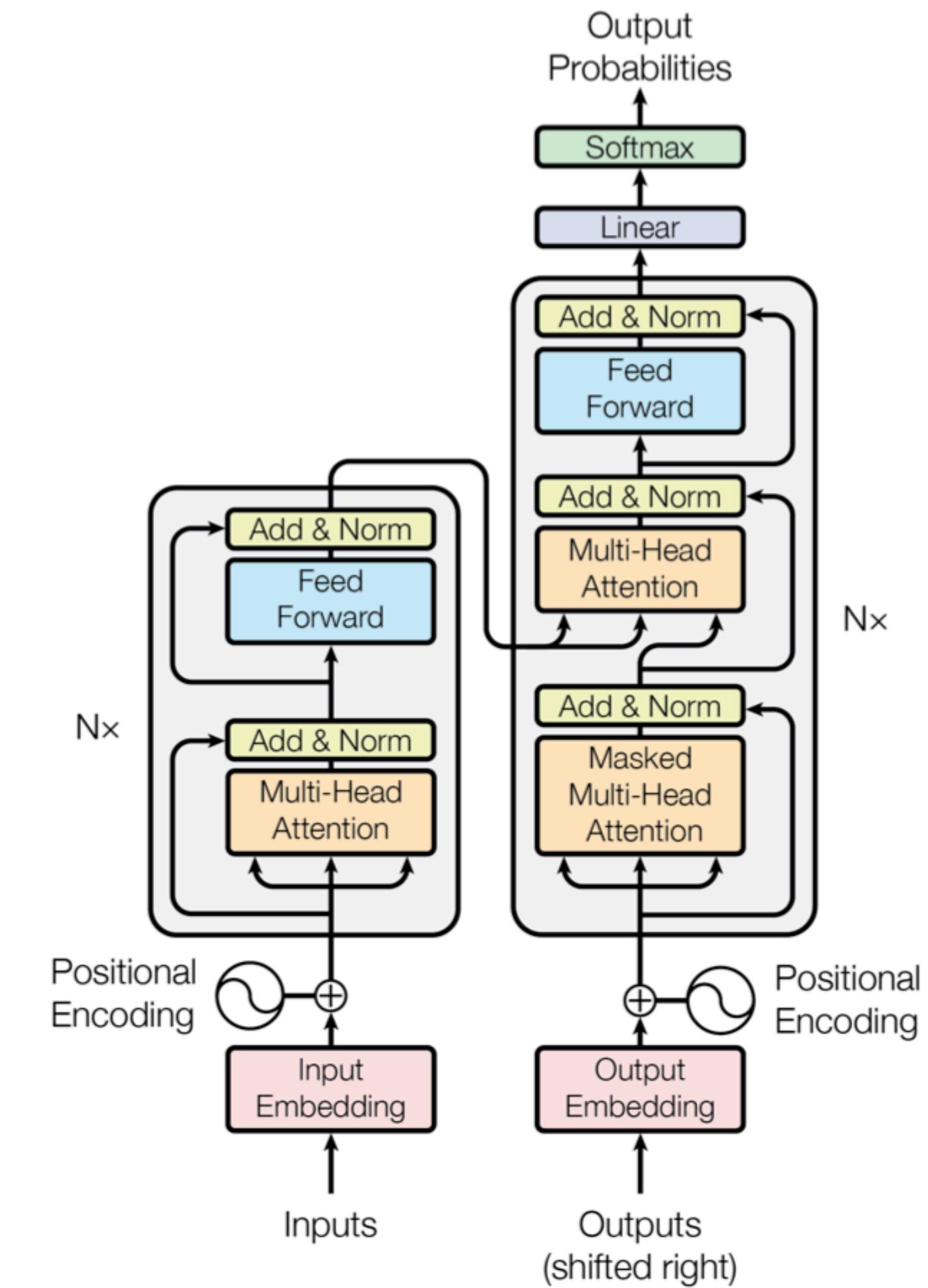
# Transformer Architecture Overview



# Transformer Architecture Overview



Let's take a closer look:  
**TRANSFORMER'S  
BUILDING BLOCKS**



# Scaled Dot-Product Attention

To compute attention we need a scoring function

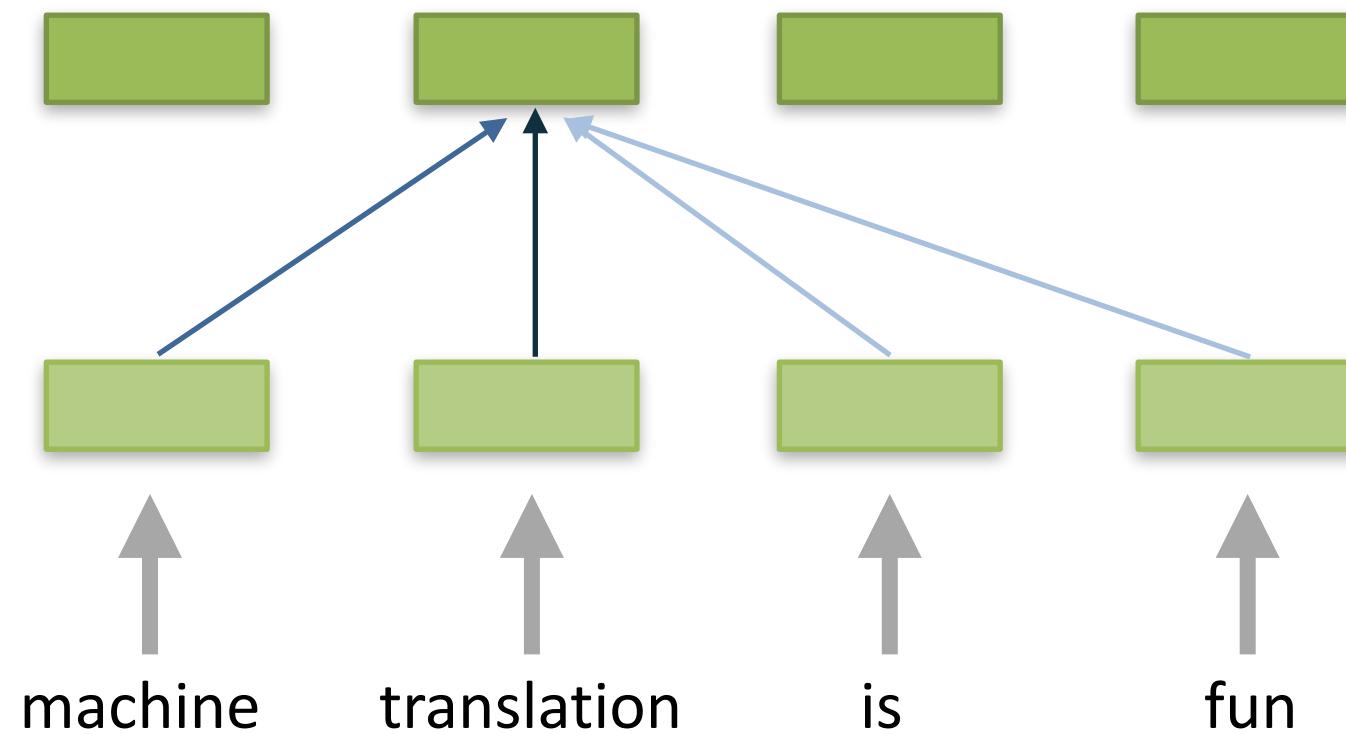
- **Dot-product** is simple and fast to compute\*
- Rationale: measure similarity of two (word-)vectors

$$\text{score}(q_t, k_i) = q_t^\top k_i$$

Problem: for high-dimensional vectors, softmax gets very peaked and gradients small

=> Solution: scale the result of dot product

$$\text{score}(q_t, k_i) = \frac{q_t^\top k_i}{\sqrt{d}}$$

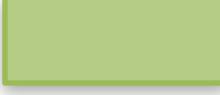


\*For a nice overview of different Attention Scoring Functions see:

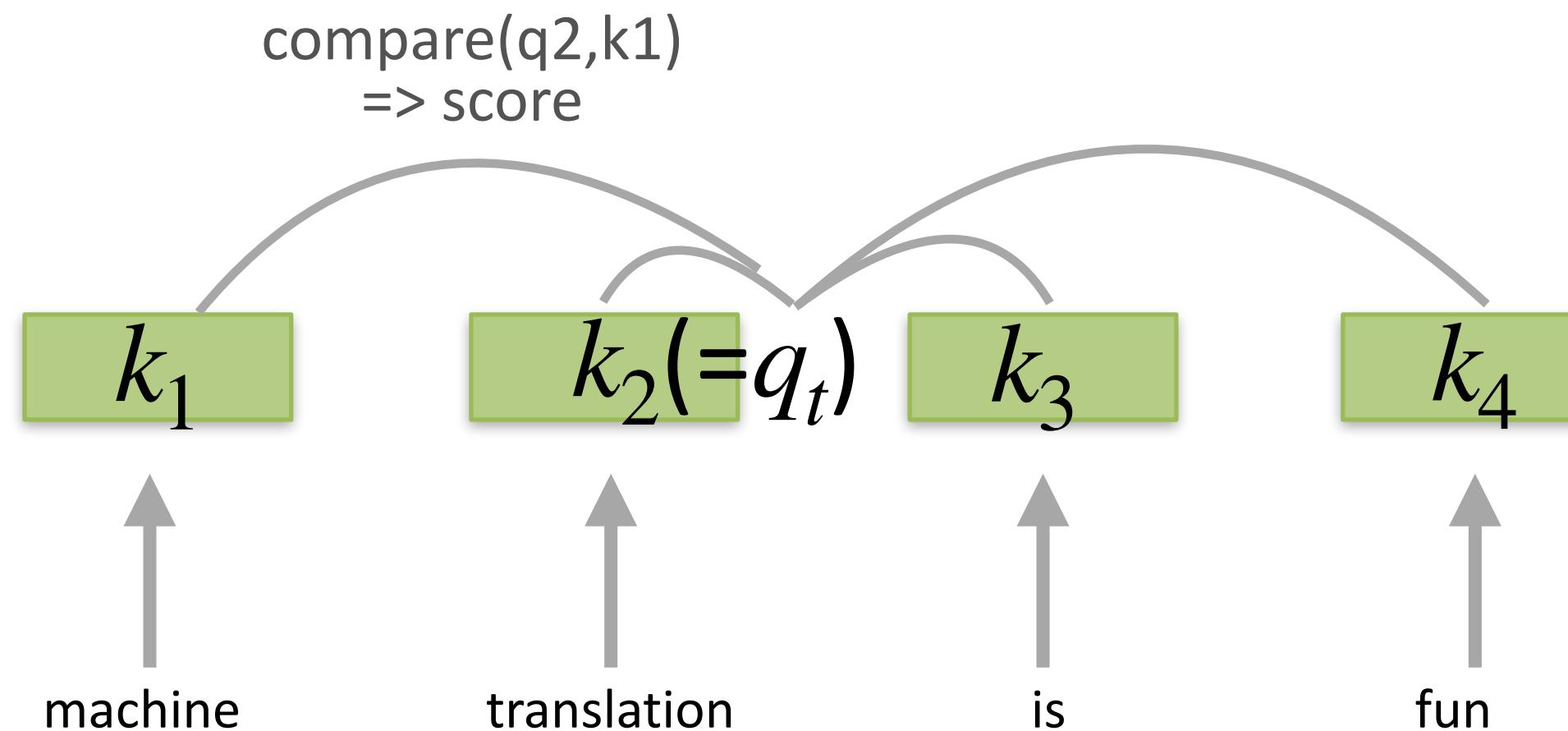
<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#ba24>

# Query-Key-Value

Now, where do  $q$  and  $k$  come from?

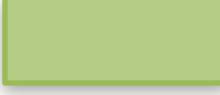
We could simply use the word vector  and compare it to all vectors in the sentence (including itself)

$$\text{score}(q_t, k_i) = \frac{q_t^\top k_i}{\sqrt{d}}$$



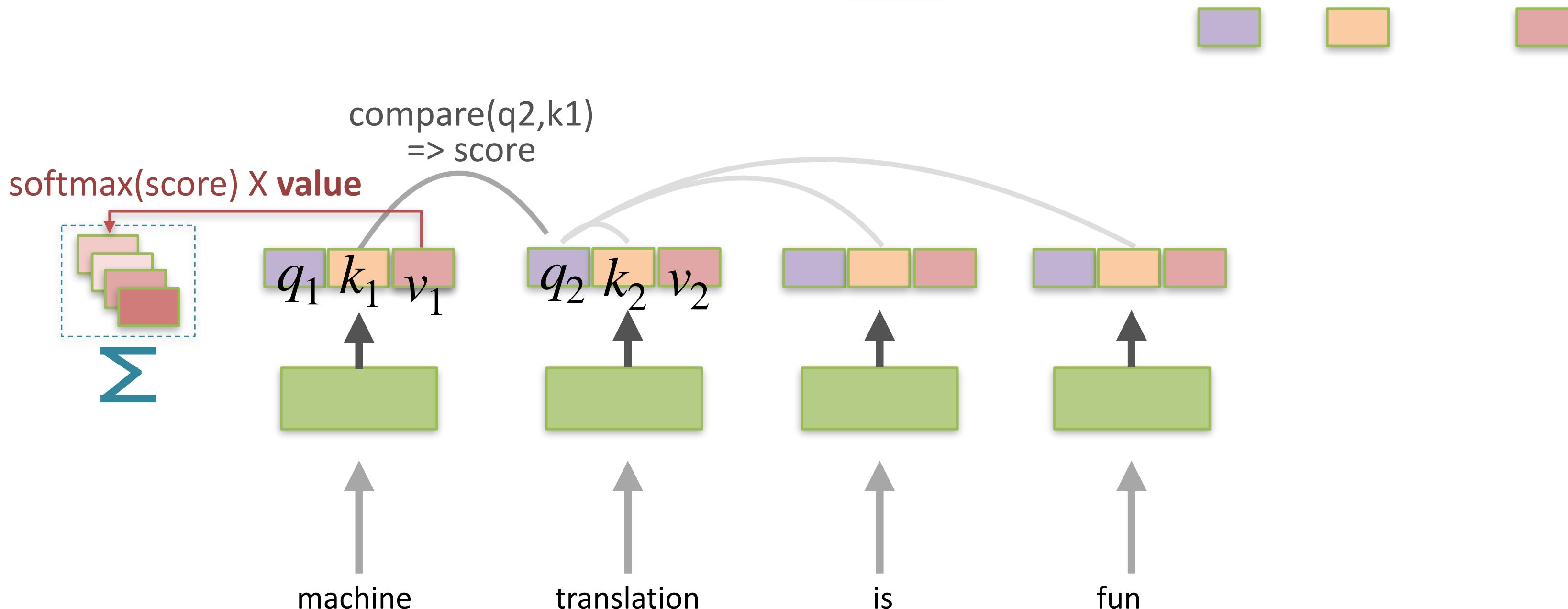
# Query-Key-Value

Now, where do  $q$  and  $k$  come from?

We could simply use the word vector  and compare it to all vectors in the sentence (including itself)

$$\text{score}(q_t, k_i) = \frac{q_t^\top k_i}{\sqrt{d}}$$

A better idea: Learn multiple ‘views’ of  to use as **query**, **key** and **value**



# Self Attention in the Transformer

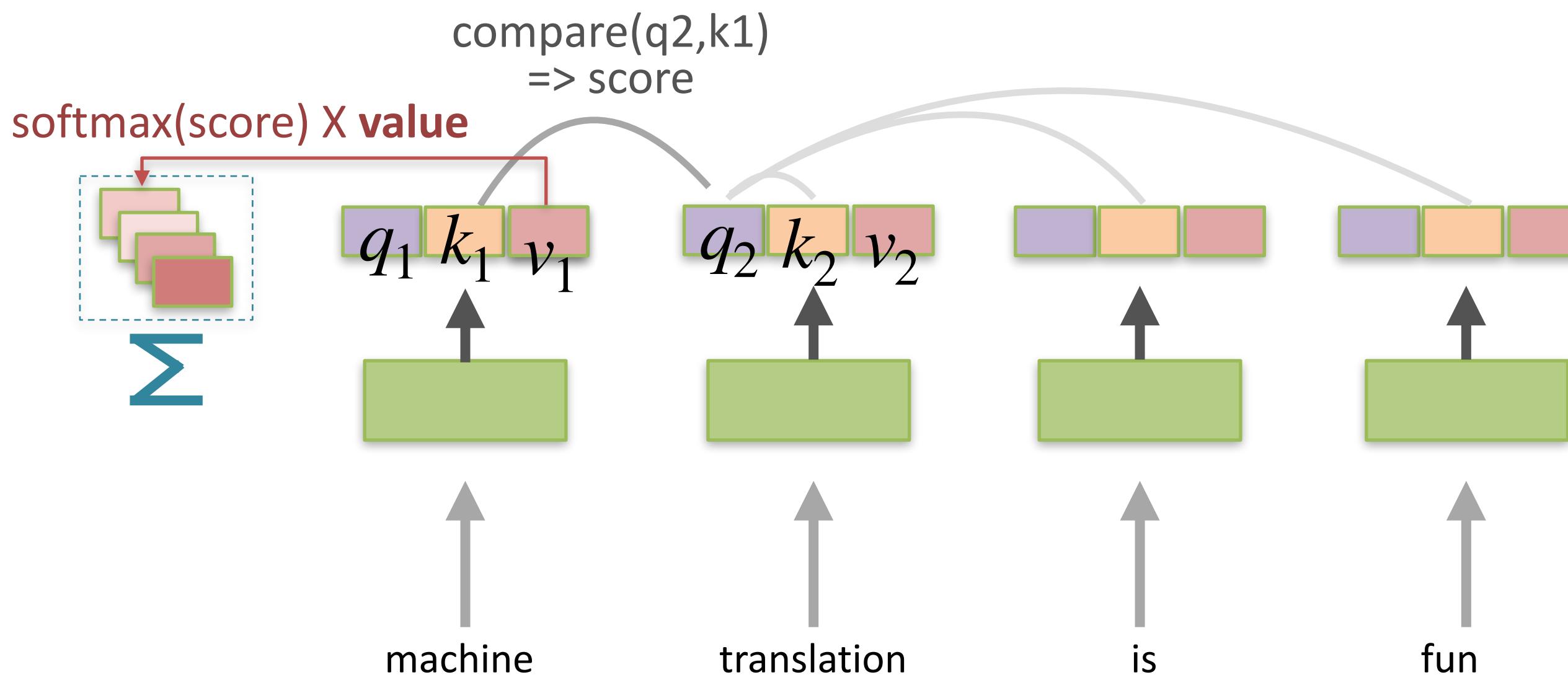
- Transform each token representation to a **learnable query**, **key**, and **value** vector
  - **Query vector** “asks” all **key vectors** in the sequence whether they are relevant. This results in **attention scores**.
  - “Asking” is the **dot product** (similarity) between the **query vector** and **key vector**
  - New **contextualised** token representation after self-attention is the **weighted average** over all **value vectors** in the sequence using attention scores as weights

# Query-Key-Value

Now, where do  $q$  and  $k$  come from?

We could simply use the word vector  and compare it to all vectors in the sentence (including itself)

A better idea: Learn multiple ‘views’ of  to use as **query**, **key** and **value**



$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{Z} = \text{softmax}(\mathbf{Q}\mathbf{K}/\sqrt{d_k})\mathbf{V}$$

**We are not done yet ...**

# Multi-Head Attention

- **Intuition:** Information from different parts of the sentence can be useful to disambiguate in different ways

I run a small business

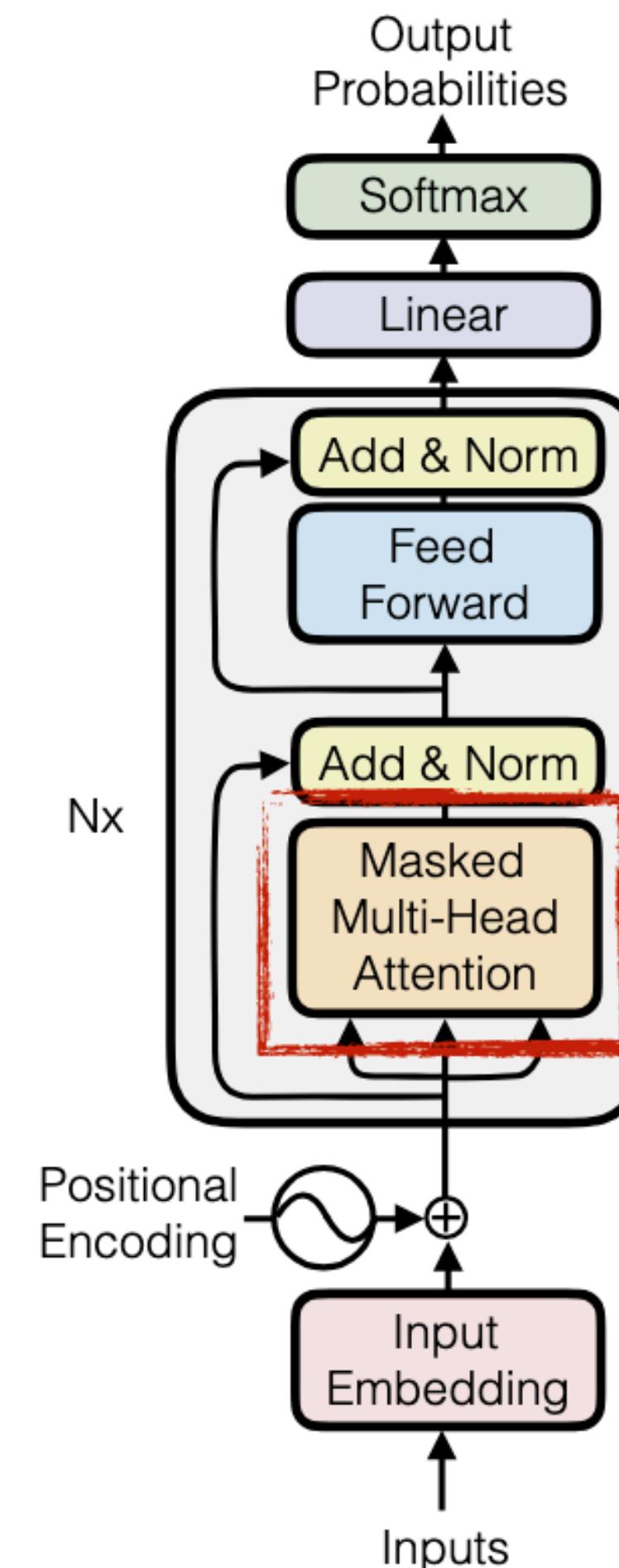
syntax  
(nearby context)

I run a mile in 10 minutes

semantics  
(farther context)

The robber made a run for it

The stocking had a run

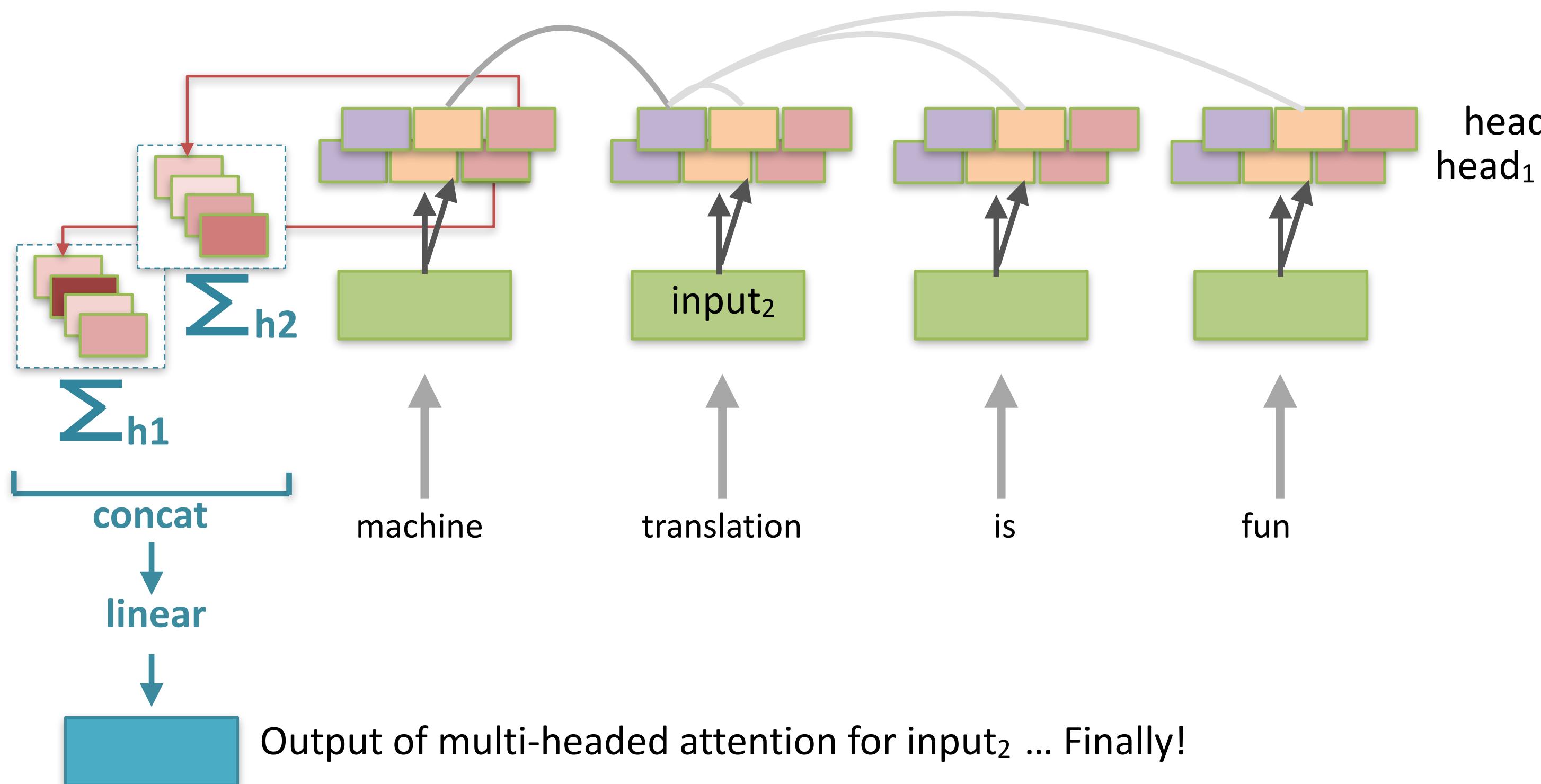


# Multi-Head Attention

Words can interact with each other in different ways.

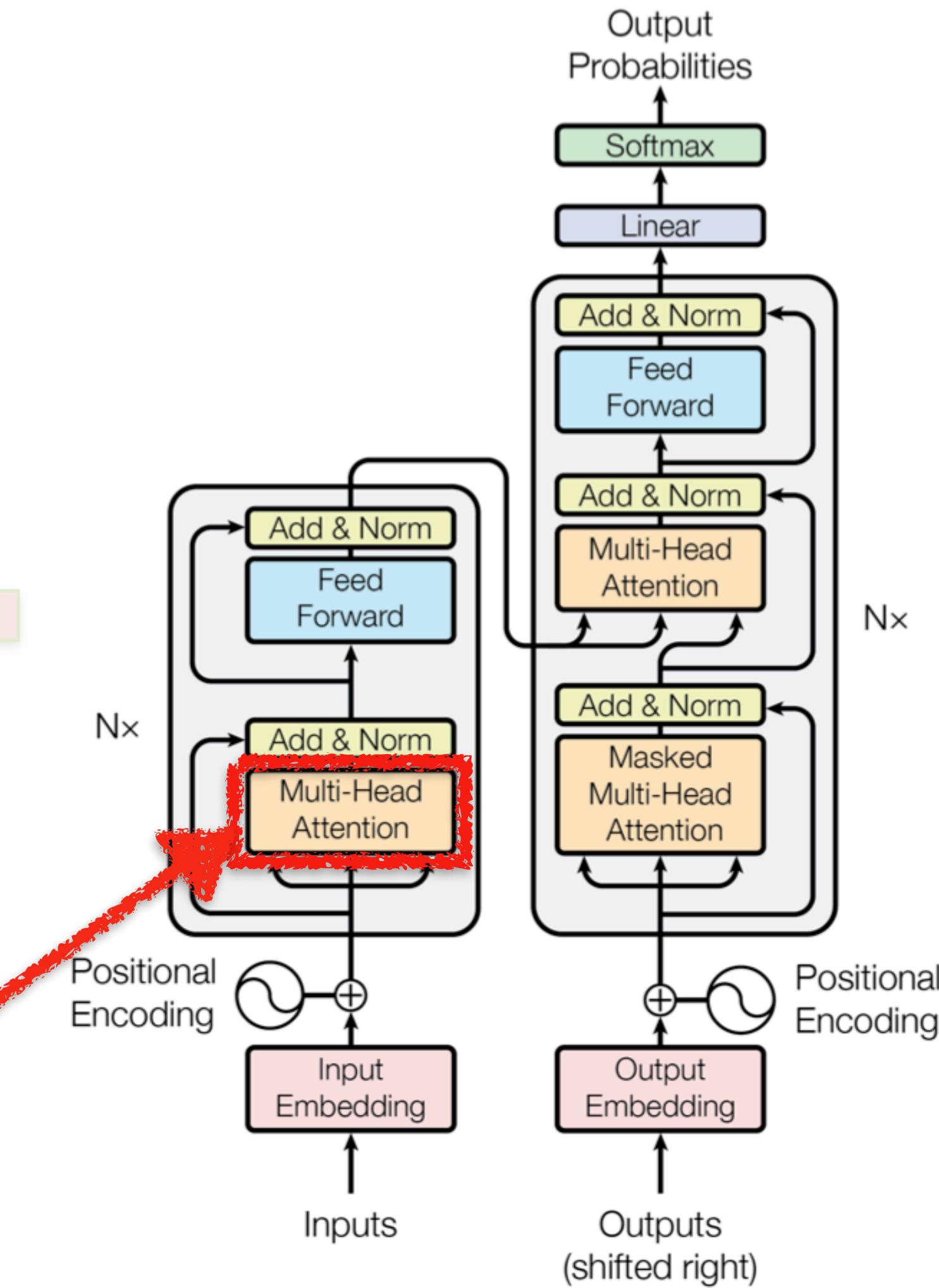
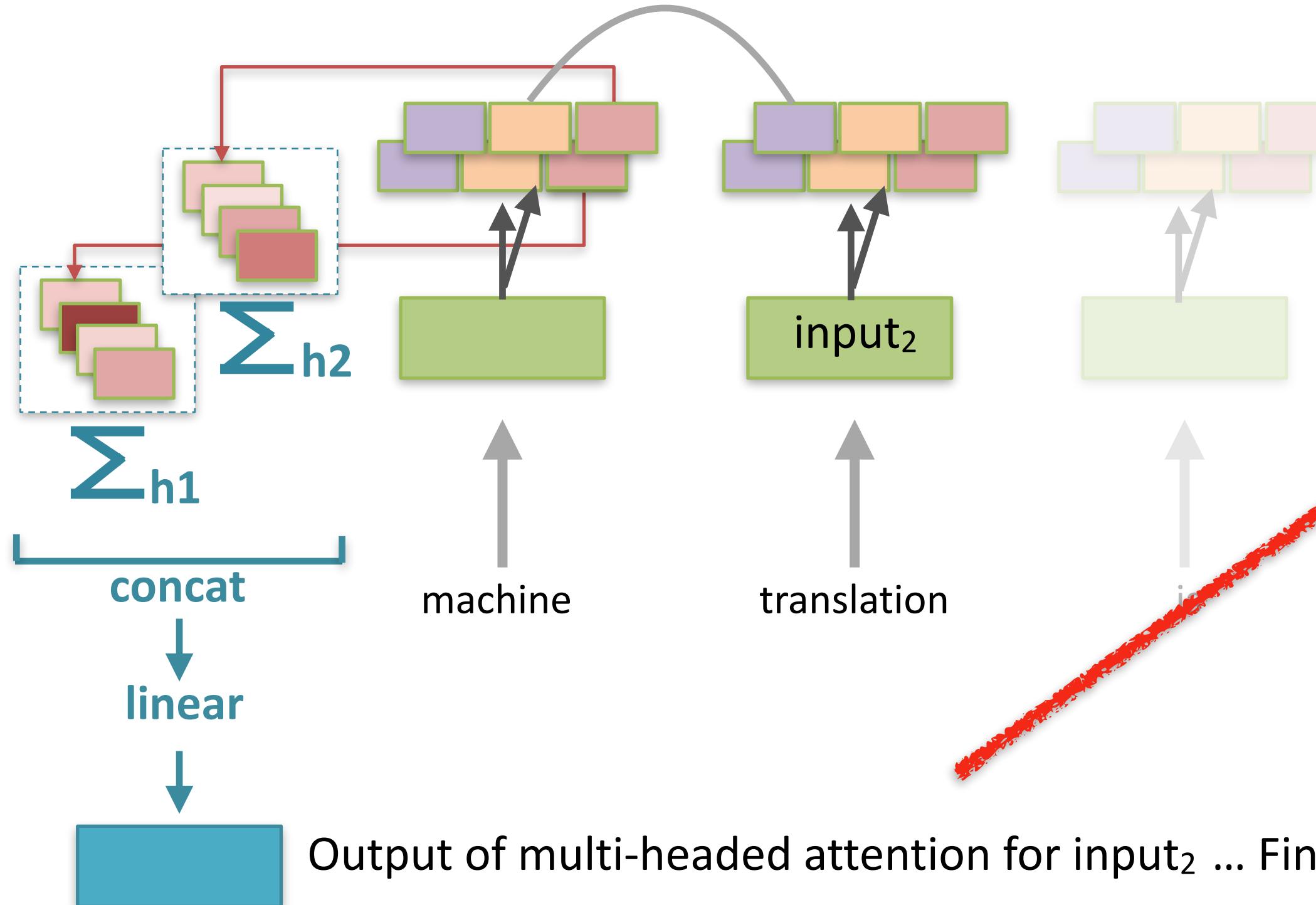
One attention distribution may not be enough to capture: coreference effects, topic cohesion, other syntactic/semantic relationships, etc.

Multi-Head gives the attention layer multiple *representation subspaces*



# Multi-Head Attention

$\text{MultiHead}(Q, K, V) = [\text{head}_1; \dots; \text{head}_h]W^O$   
 where  $\text{head}_i = \text{Attention}(\hat{Q}_i, \hat{K}_i, \hat{V}_i)$   
 $= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$



# Multi-Head Attention

# Example from Vaswani et al., 2017:

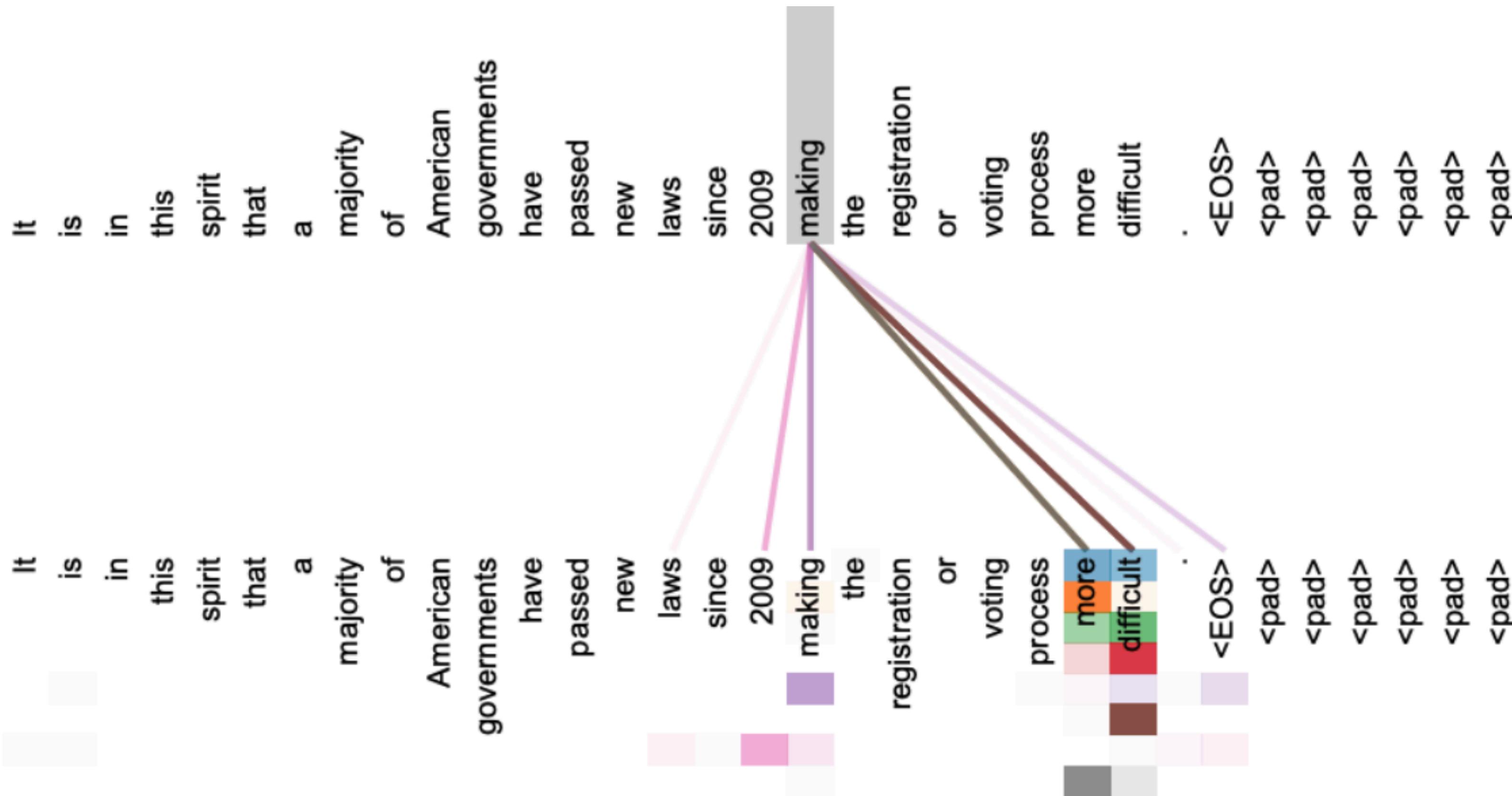
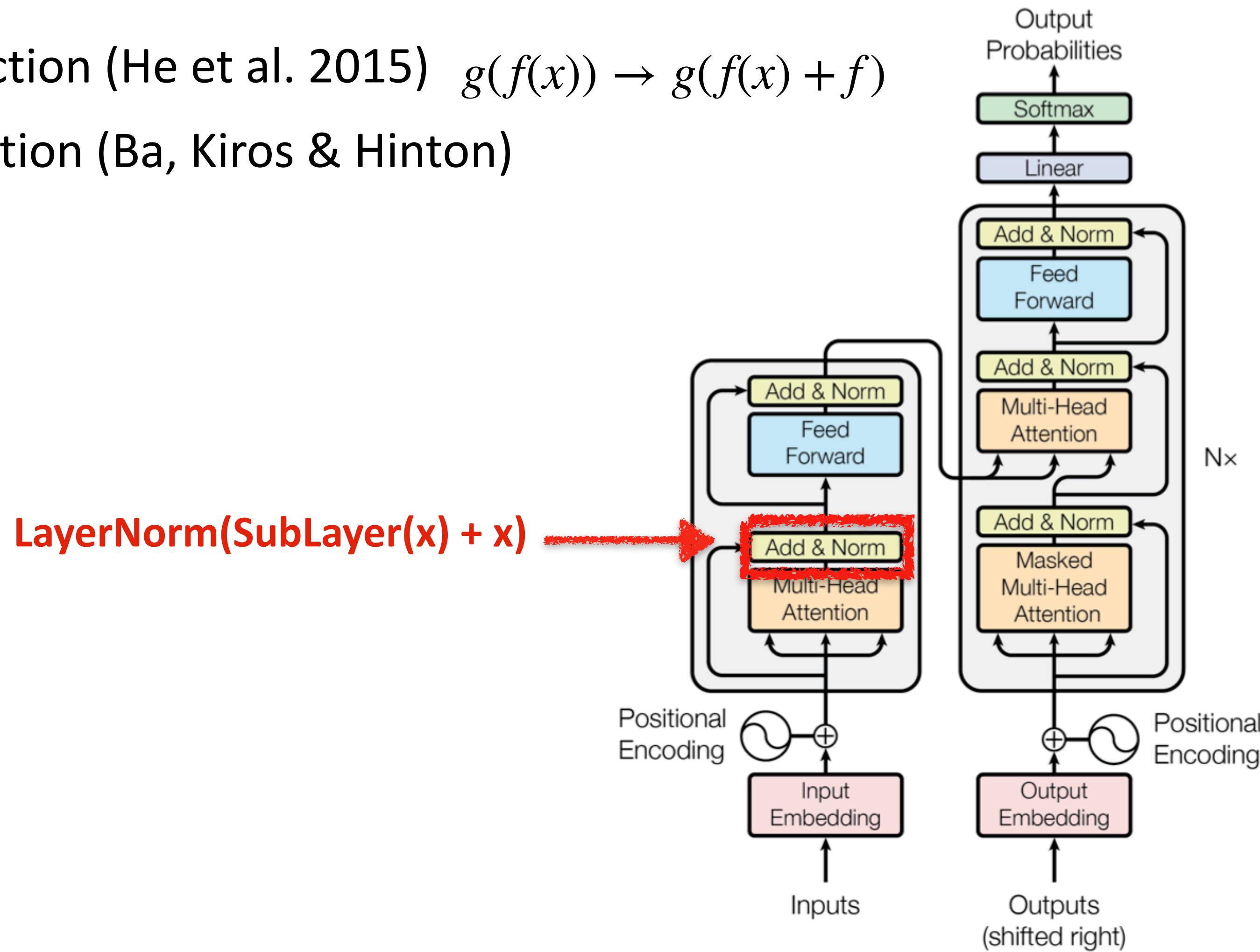


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

# Add & Norm

Last ingredients:

- Residual connection (He et al. 2015)  $g(f(x)) \rightarrow g(f(x) + f)$
- Layer normalization (Ba, Kiros & Hinton)



# Attention in the Decoder

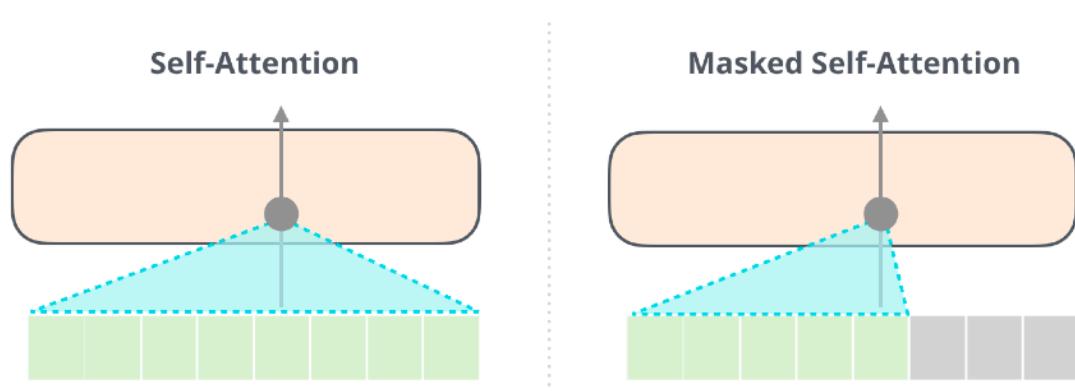
We have looked at self-attention in the encoder

Now moving to the decoder => 2 types of attention here:

1

Masked Self Attention (Decoder only):

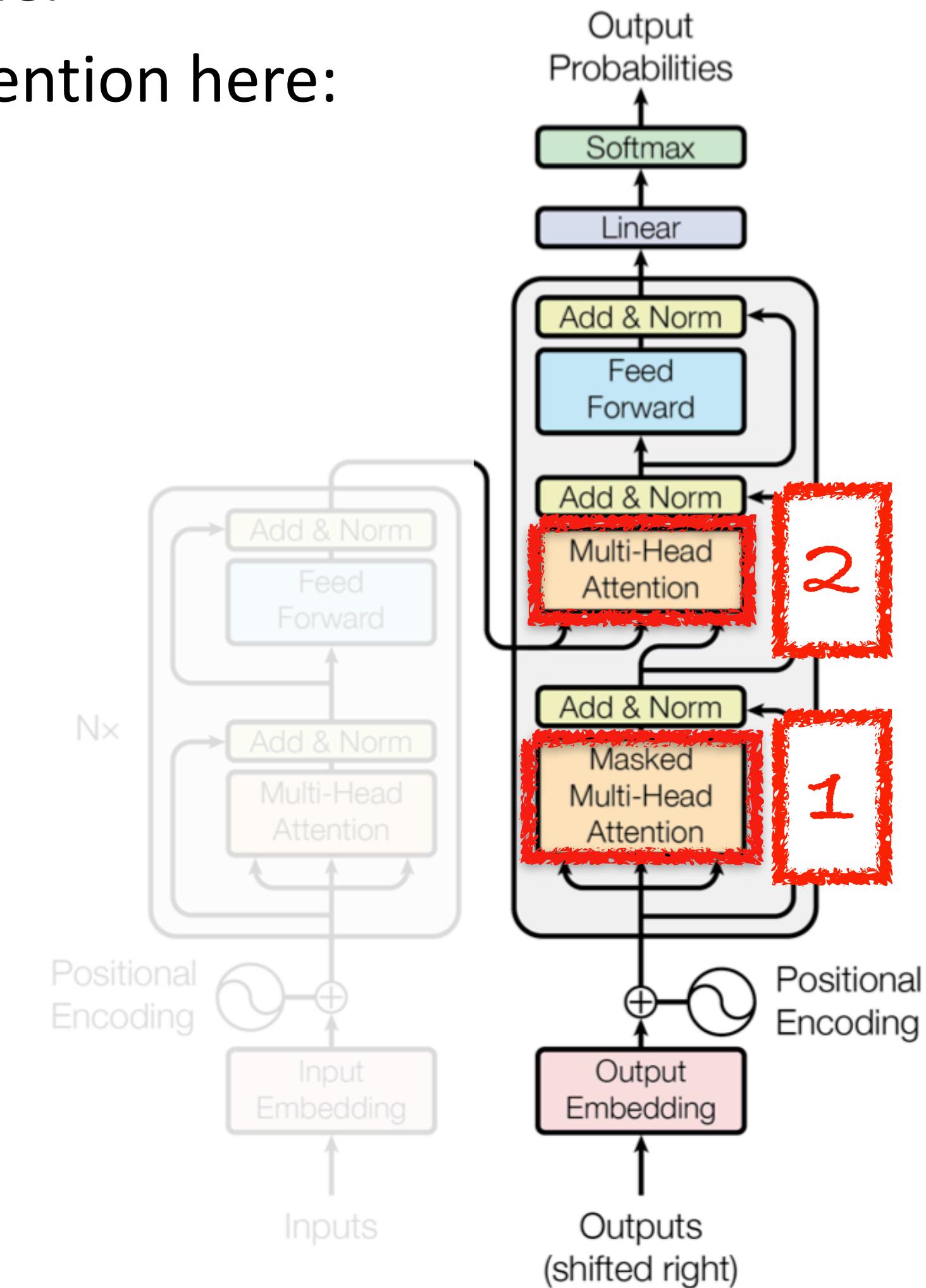
- captures target-side context
- same as before, but can only look at positions before the current word (*masked*)



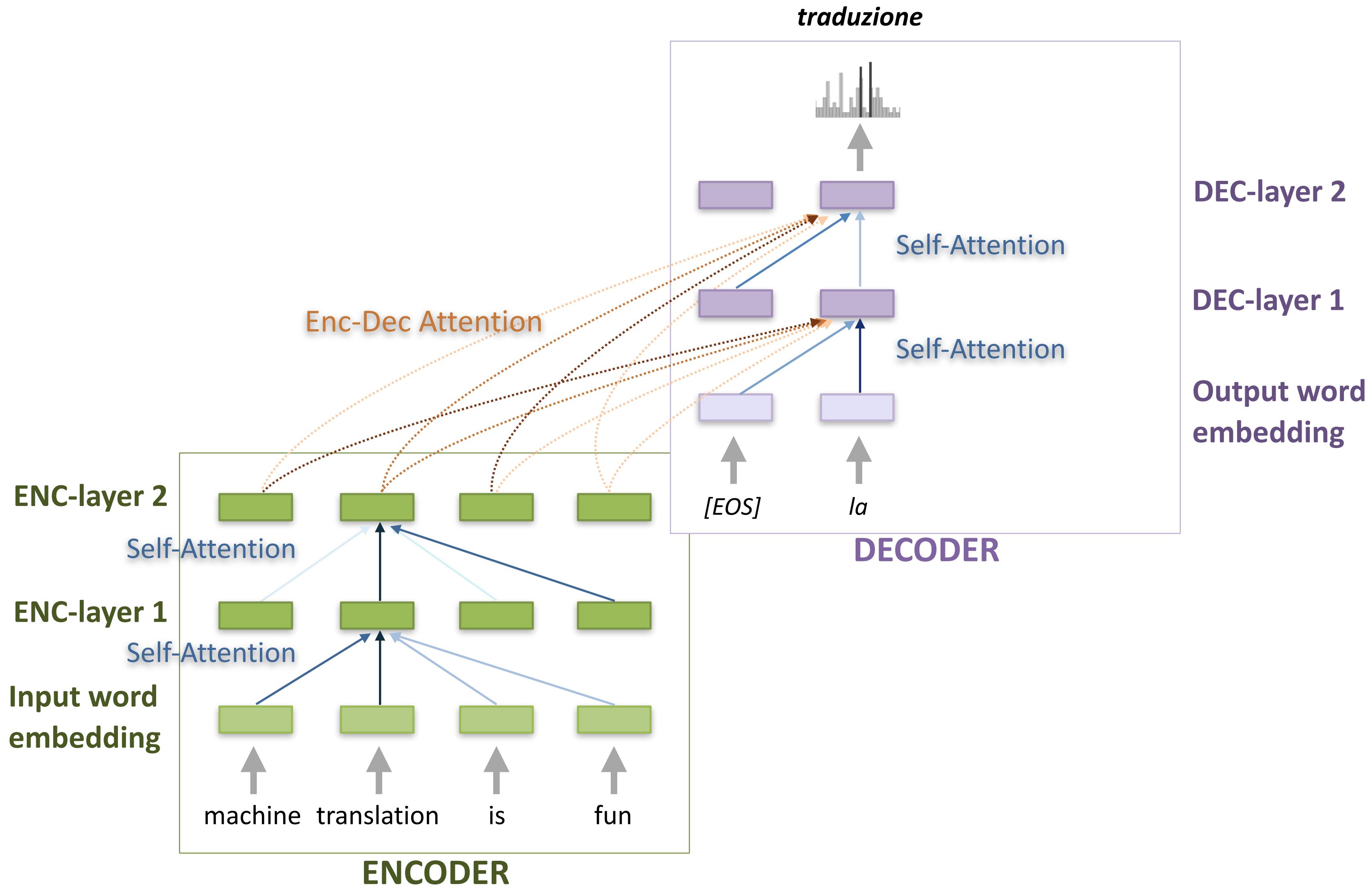
2

Encoder→Decoder Attention:

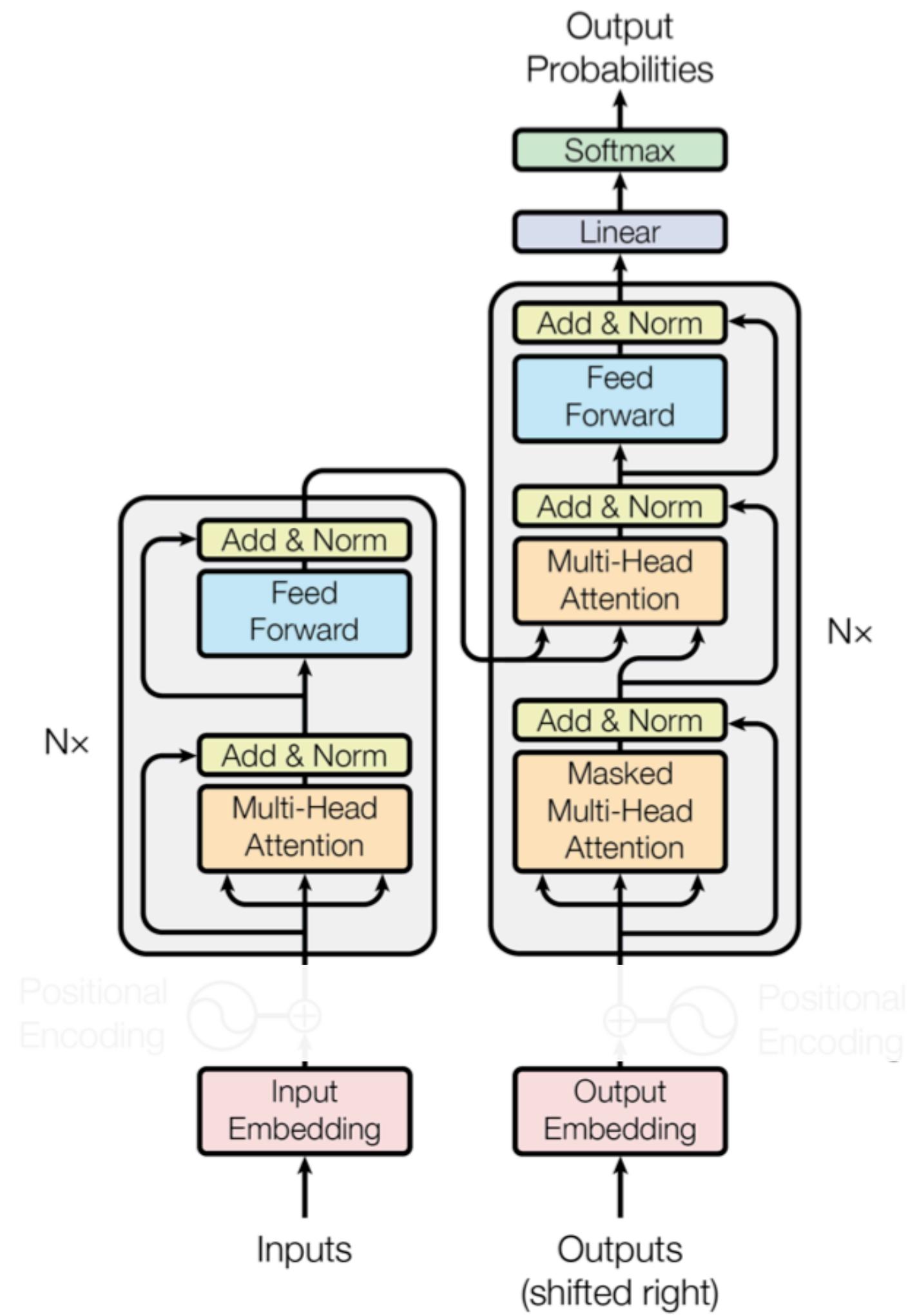
- captures src-trg translation equivalences
- Query comes from target (decoder), Key & Value from source (encoder)
- also known as Cross Attention



# Transformer Architecture Overview



Are we missing anything?



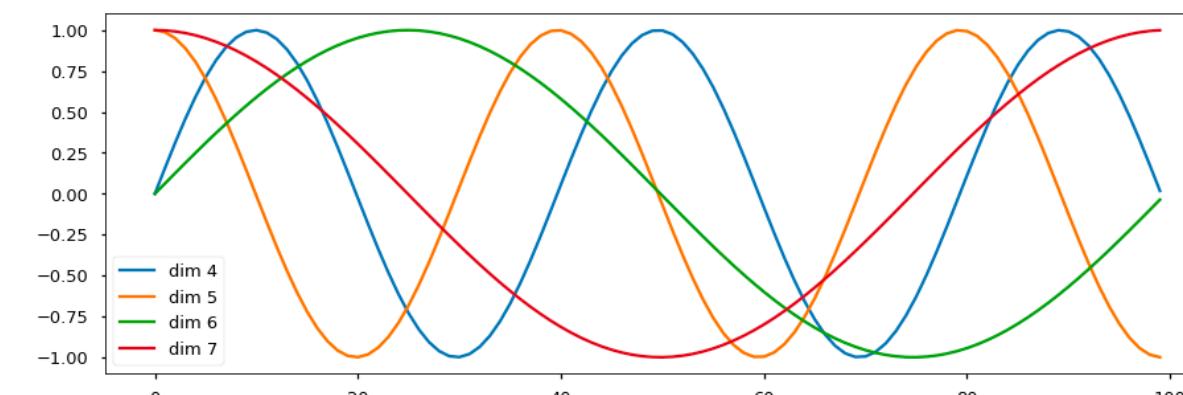
# Positional embeddings

Recurrency naturally represents the order of words in a sentence:

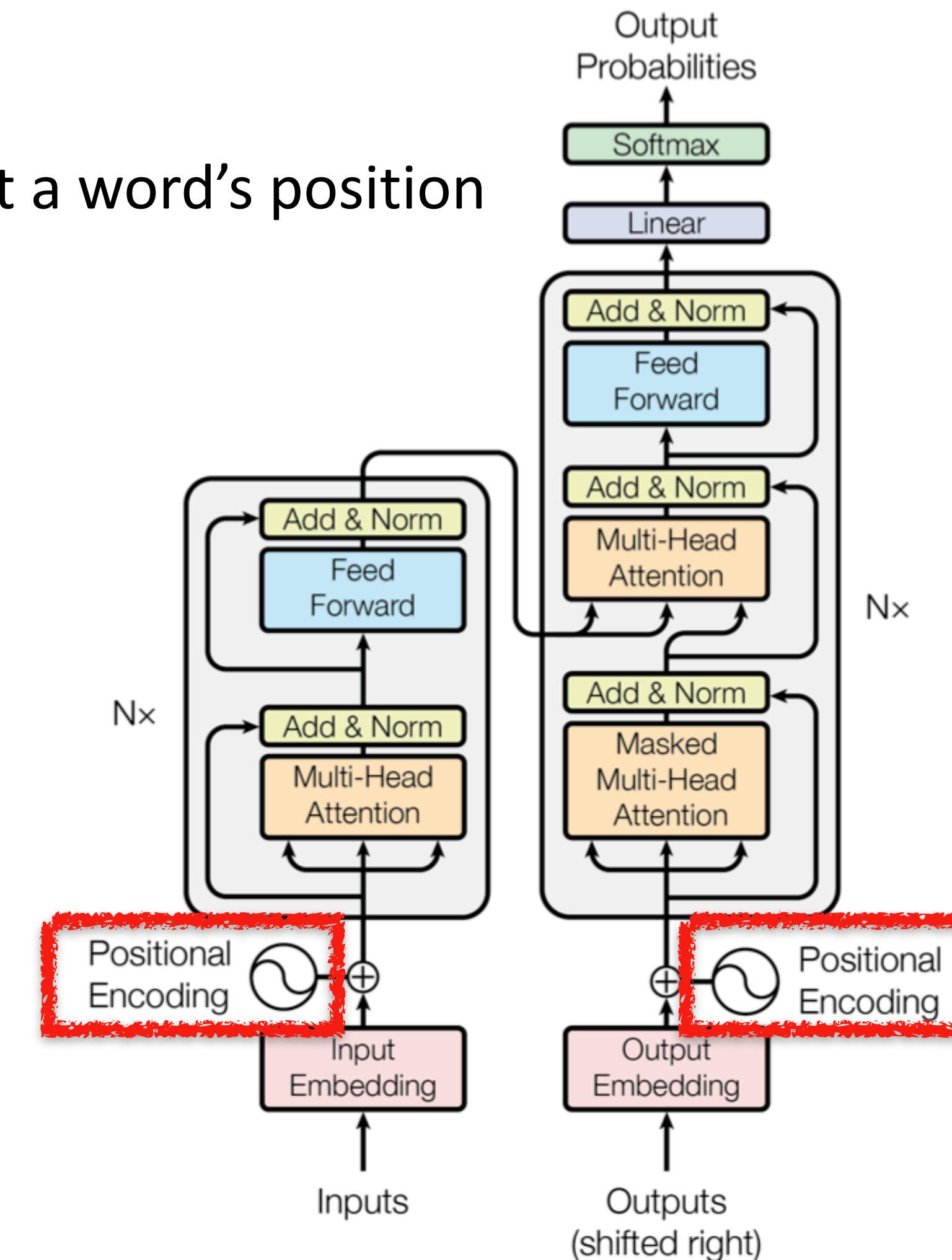
*w<sub>3</sub> comes after w<sub>2</sub> which comes after w<sub>1</sub> ...*

Transformer needs an explicit way to represent a word's position

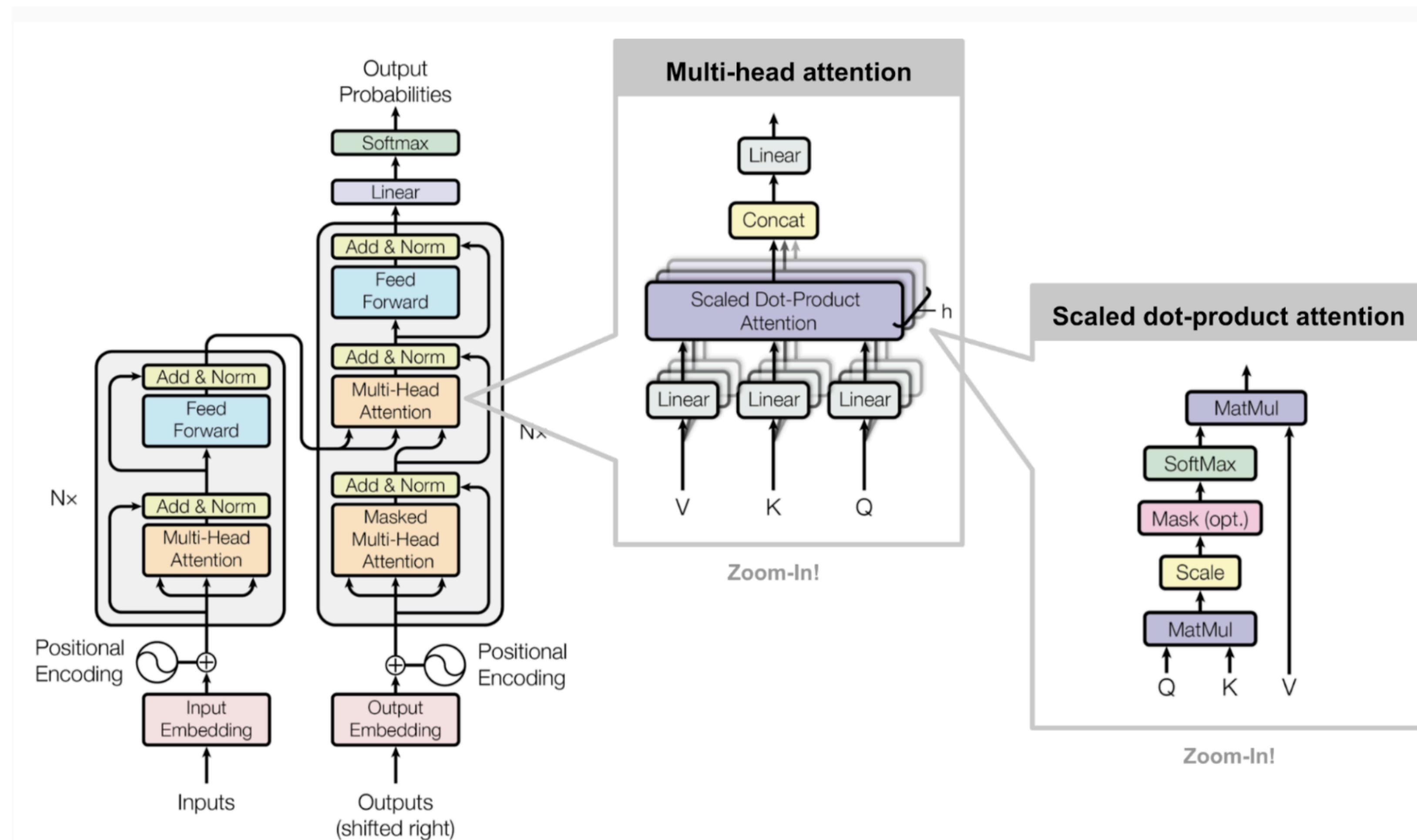
- Idea: let's mark positions
- Learning will figure out what how to use them
- Simple version: **learnable** embeddings  $\phi_p(i)$  where  $i \in [0,1,2,\dots]$  (lookup parameters)
- More advanced: **fixed** embeddings, where values determined by sine waves, with different frequency and offset of each dimensions



- Either way, we add them to token embeddings



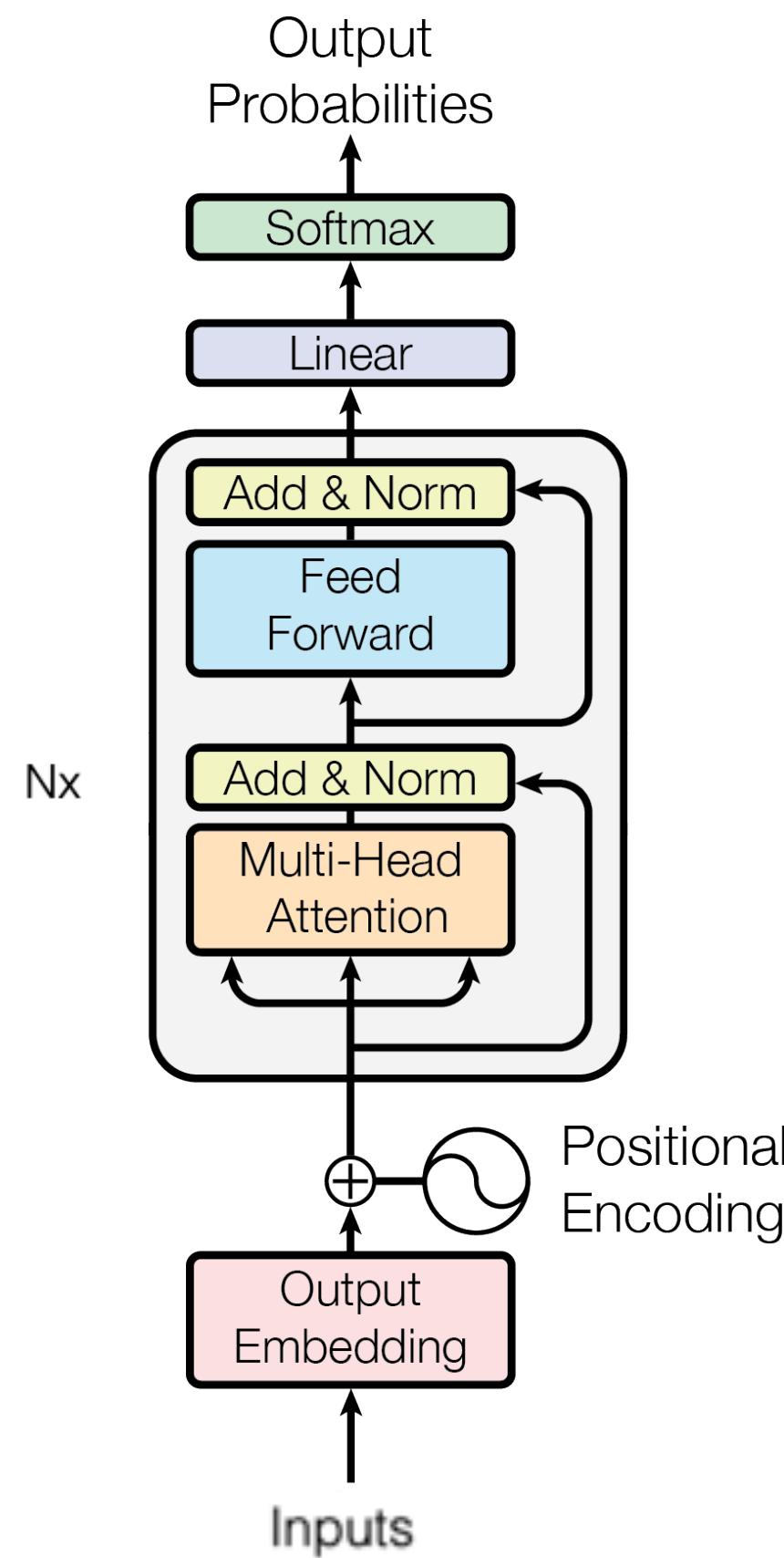
# Putting it altogether: Encoder-Decoder Transformer



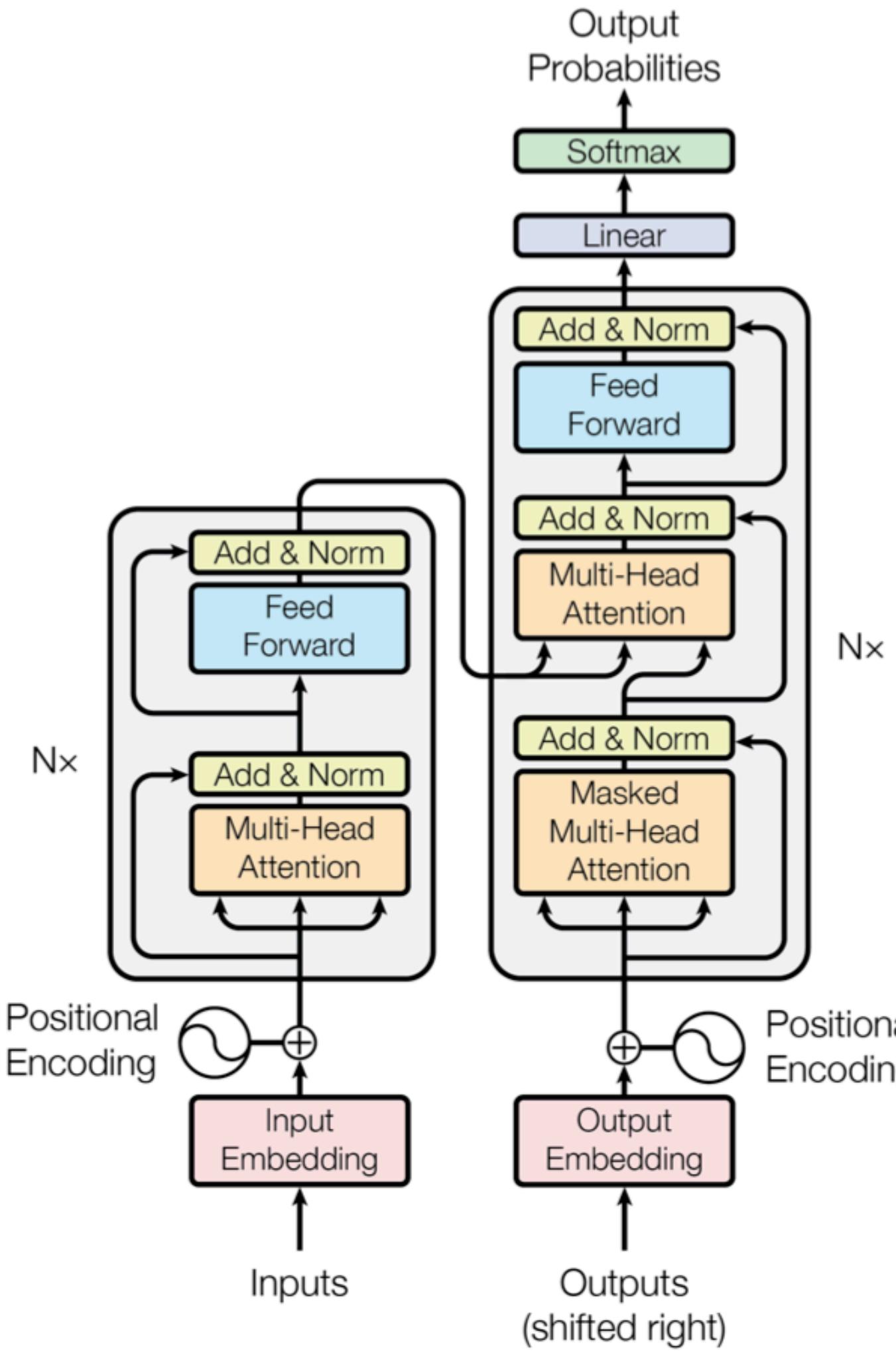
# Transformer: Summary of Core Concepts

- ▶ Attention:
  - ▶ Self Attention
  - ▶ Cross Attention (Encoder-Decoder Attention)
  - ▶ Multi-Head Attention
  - ▶ Masked Attention
- ▶ Positional encodings
- ▶ Residual connections + layer normalisation
- ▶ Feed-forward layer

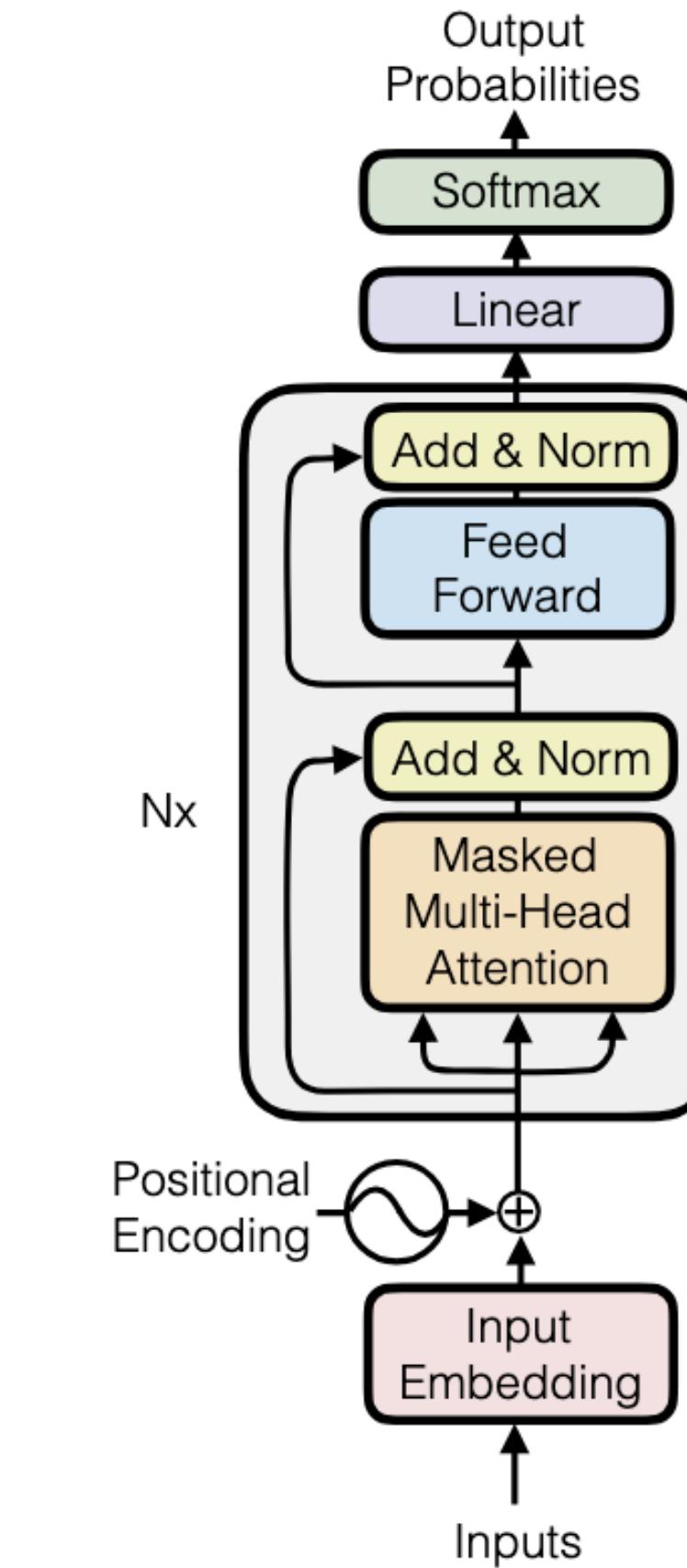
# Three Types of Transformers



**Encoder-Only Model**  
(e.g. BERT)



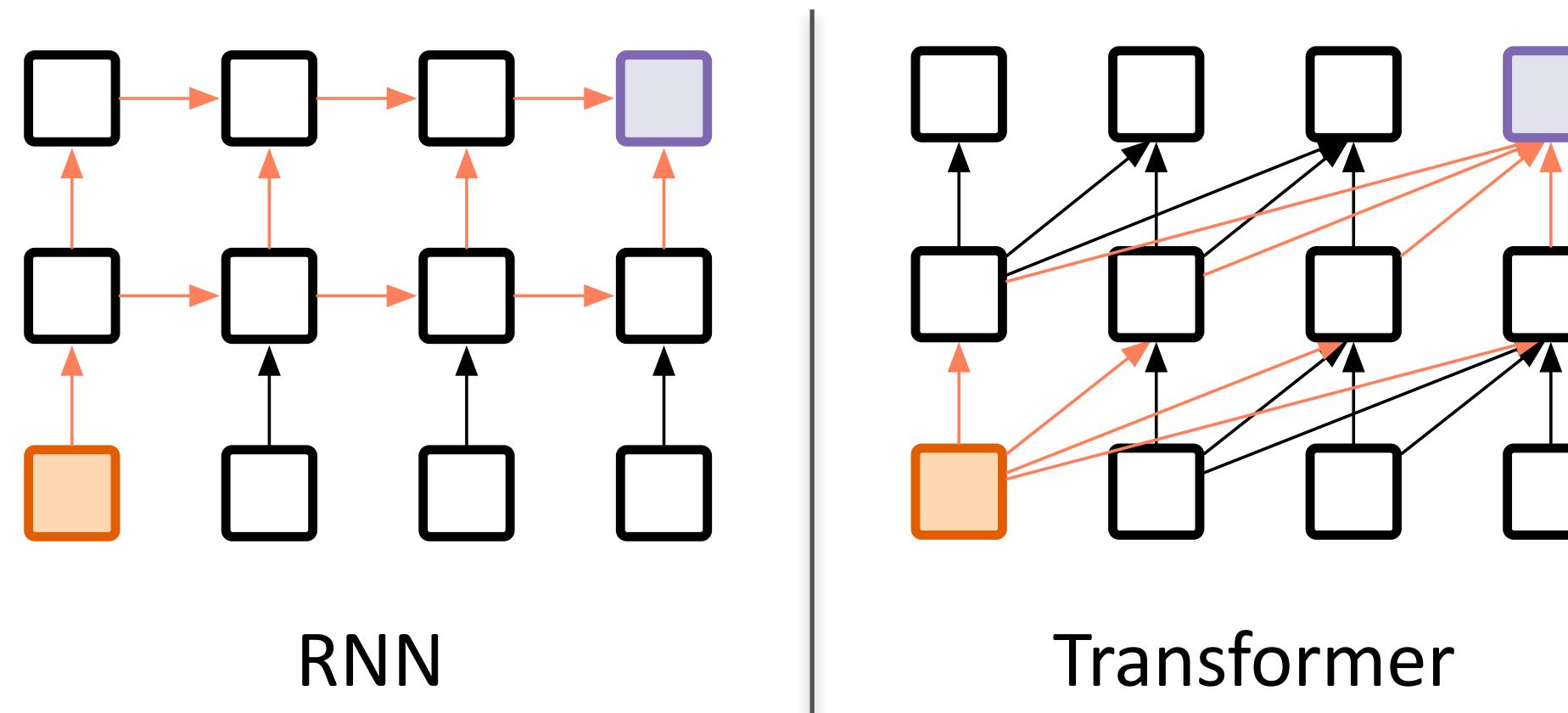
**Encoder-Decoder Model**  
(Vaswani et al., 2017), T5



**Decoder-Only Model**  
(e.g. GPT, Llama)

# **RECURRENT SEQ-TO-SEQ VS TRANSFORMER**

# RNN-seq2seq vs Transformer



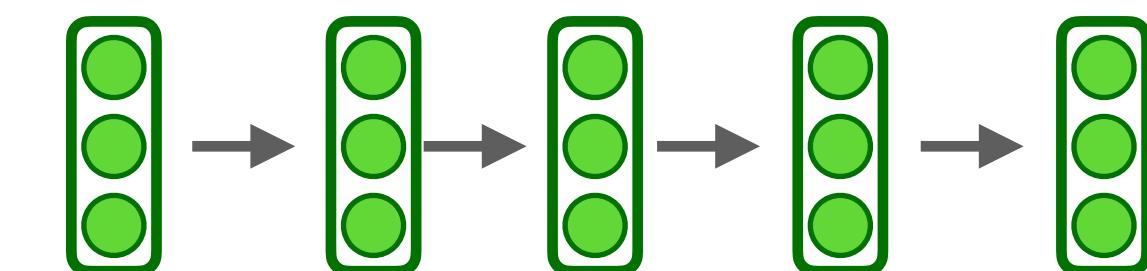
- ✓ Much more parallelizable  
= fast
- ✓ Lower complexity
- ✓ Shorter path among any  
input positions



- RNNs (esp. LSTM) are cognitively inspired: represent memory constraints
- Transformer = result of clever engineering & brute-force architecture search
  - It works! State of the art performance on almost every NLP task
- Will we go back to RNN-kind of models?

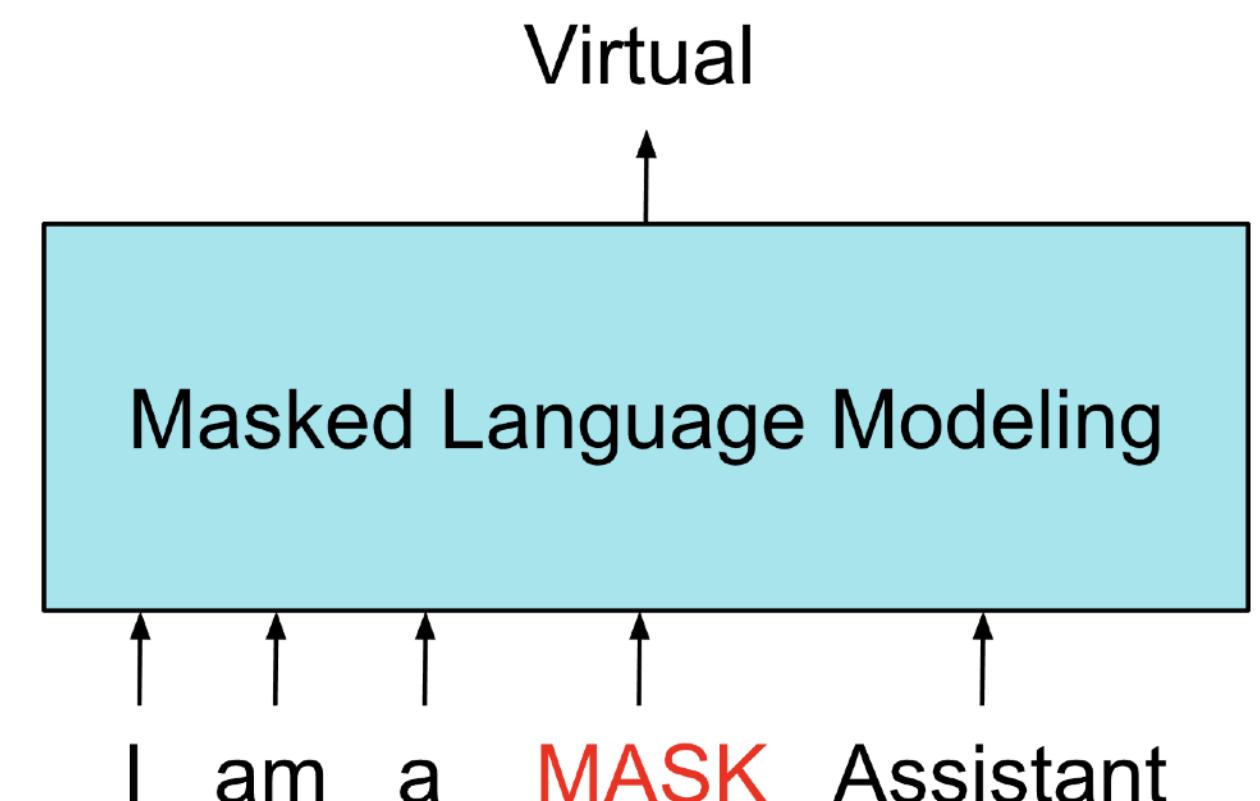
# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT) Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



# Masked LMs: The birth of BERT

- LMs so far: predict the next token given the previous tokens
  - This enables a self-supervised task
  - That we can train on a lot of data to get really useful representations
- Let's **create a prediction task** by **hiding** part of the sequence, and then trying to predict them
  - Input: the sequence  $\bar{x}^M$  where some tokens are replaced with the token [MASK], for example:  
$$\bar{x}^M = \langle x_1, \dots, x_4, [\text{MASK}], x_6, \dots, x_n \rangle$$



We thank the following sources for the materials on which slides on this section are based: Greg Durrett & Yoav Arzti

Image from <https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering>

# BERT

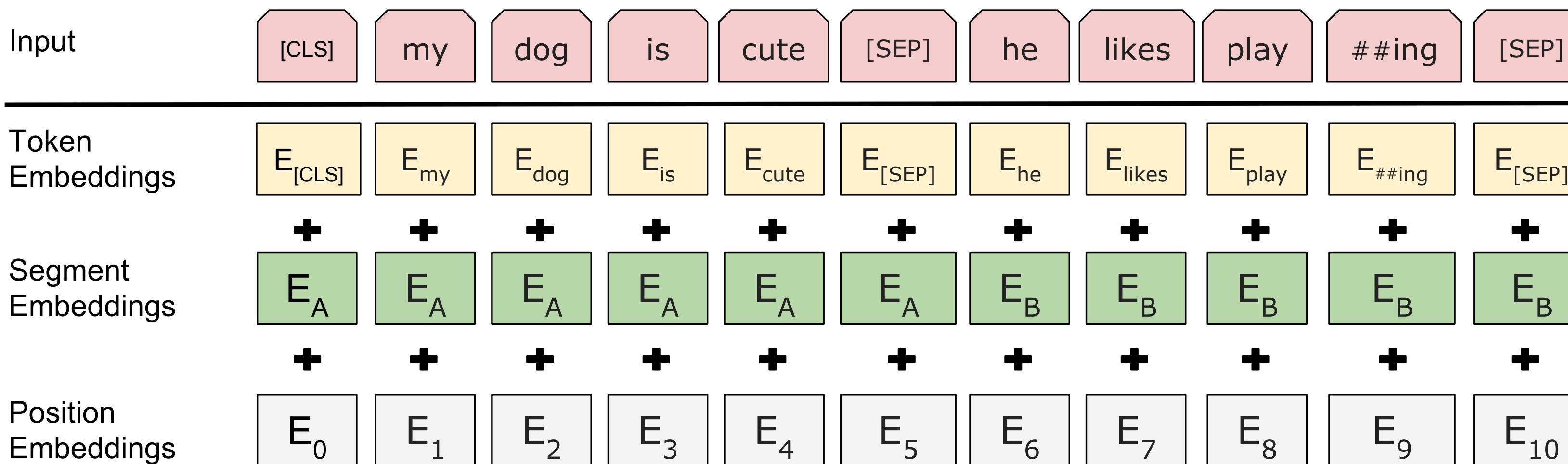
## Bidirectional Encoder Representations from Transformers

- Encoder transformer
  - Encoders assume we have the complete sequence
  - No generation, only want good representations (for analysis tasks)
- BERT Base: 12 transformer blocks, 768-dim word-piece tokens, 12 self-attention heads → 110M parameters
- BERT Large: 24 transformer blocks, 1024-dim word-piece tokens, 16 self-attention heads → 340M parameters
- RoBERTa: same model, much more data (160GB of data instead of 16GB)

# BERT

## Inputs

- One or two sentences
  - Word-piece token embeddings
  - Position and segment embeddings



# BERT

## Training

- Data: raw text
- Two objectives:
  - Masked LM
  - Next-sentence prediction
- Later development in RoBERTa:
  - More data, no next-sentence prediction, dynamic masking

# BERT

## Masking Recipe for Training

- Mask and predict 15% of the tokens
  - For 80% (of 15%) replace with the input token with [MASK]
  - For 10%, replace with a random token
  - For 10%, keep the same

# BERT

## Next-sentence Prediction

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- Training data: 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk
- Predict whether the next chunk is the true next chunk
- Prediction is done on the [CLS] output representation

# BERT

## What Do We Get?

- We can feed complete sentences to BERT
- For each token, we get a contextualized token representation
  - Similar to ELMo, but without an RNN/LSTM
  - In contrast to word2vec representations that are fixed and do not depend on context
- While word2vec vectors are forced to mix multiple senses, BERT/ELMo can provide more instance-specific vectors
- BERT started an arms race towards bigger and bigger models, which quickly led to the LLMs of today

# BERT

## What It Is Not Great For?

- BERT cannot generate text
  - Not an autoregressive model, but can do filling of [MASK]
- Masked language models are intended to be used primarily for “analysis” tasks

# BERT

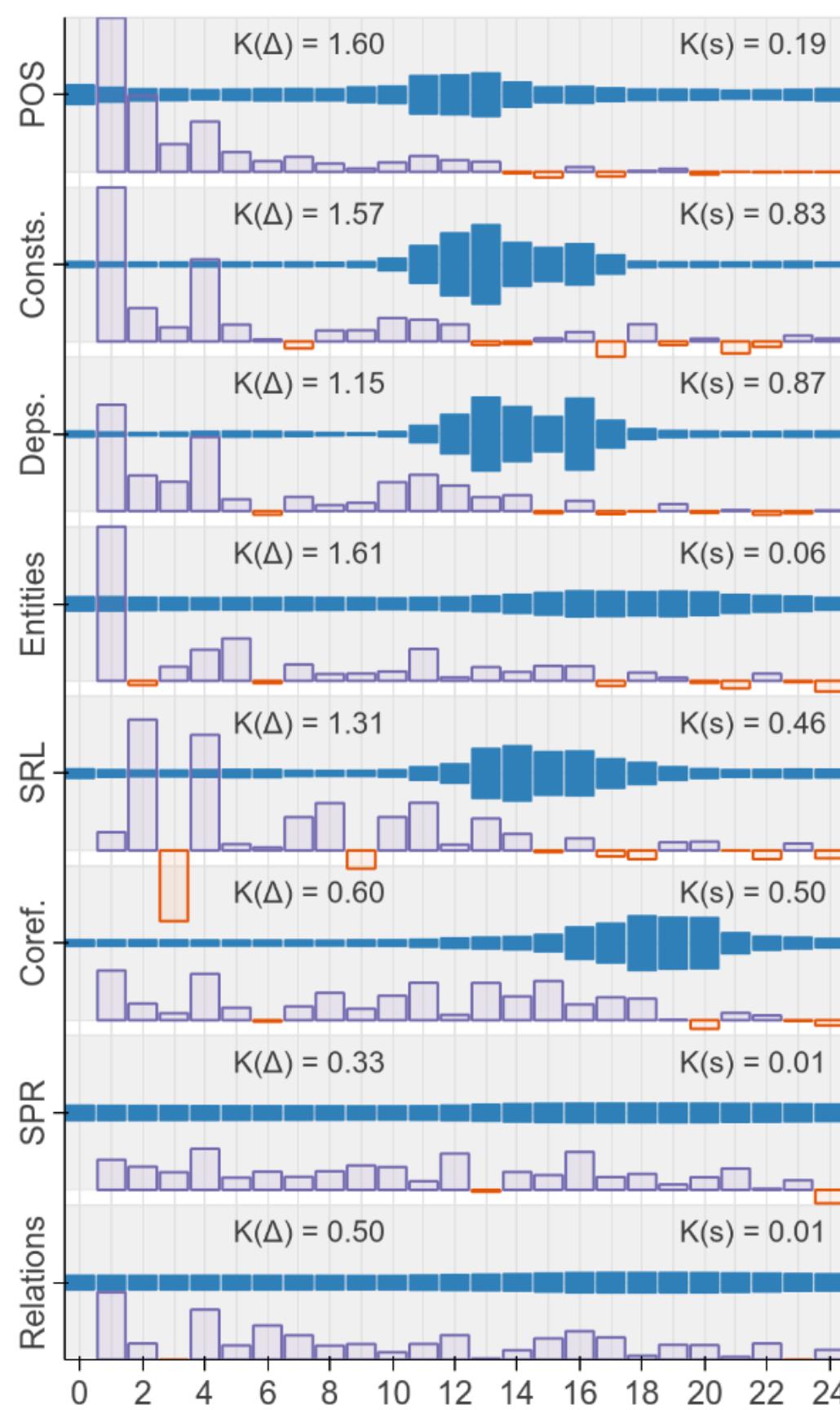
## What does BERT Learn?

- There is a lot of work trying to decipher what BERT learns in its representations
  - Much harder with recent LLMs because they are not as open

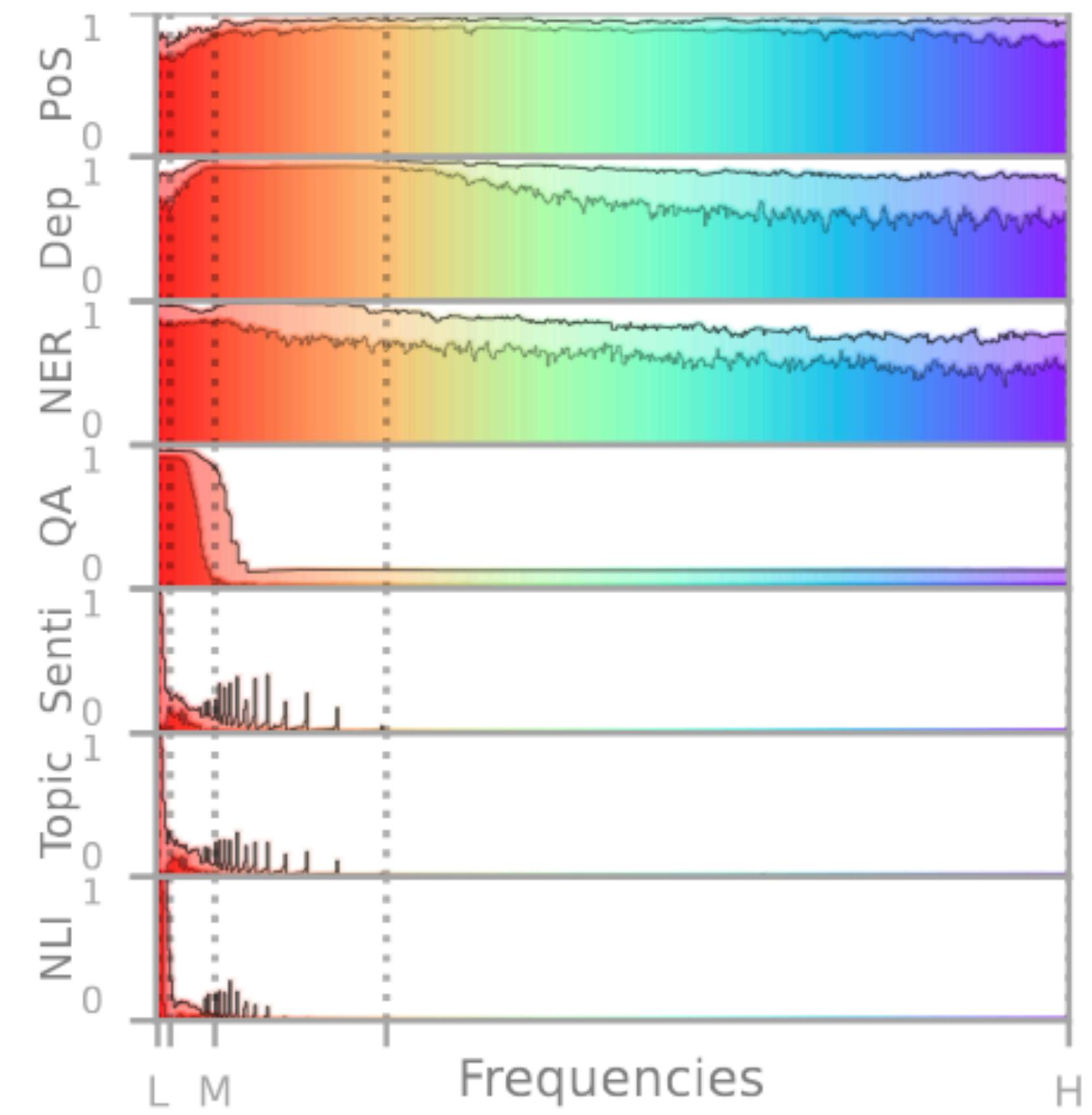
# BERT

## Some interesting results

- BERT rediscovers the classical NLP pipeline  
(Tenney et al., 2019)

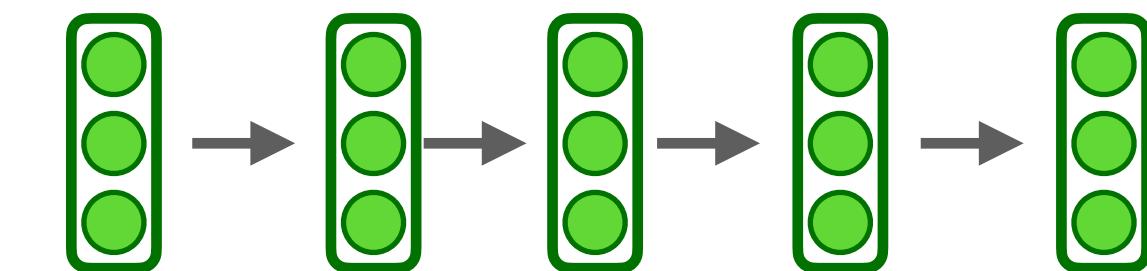


- Spectral profiles  
(Müller-Eberstein et al., 2022)



# Today's roadmap

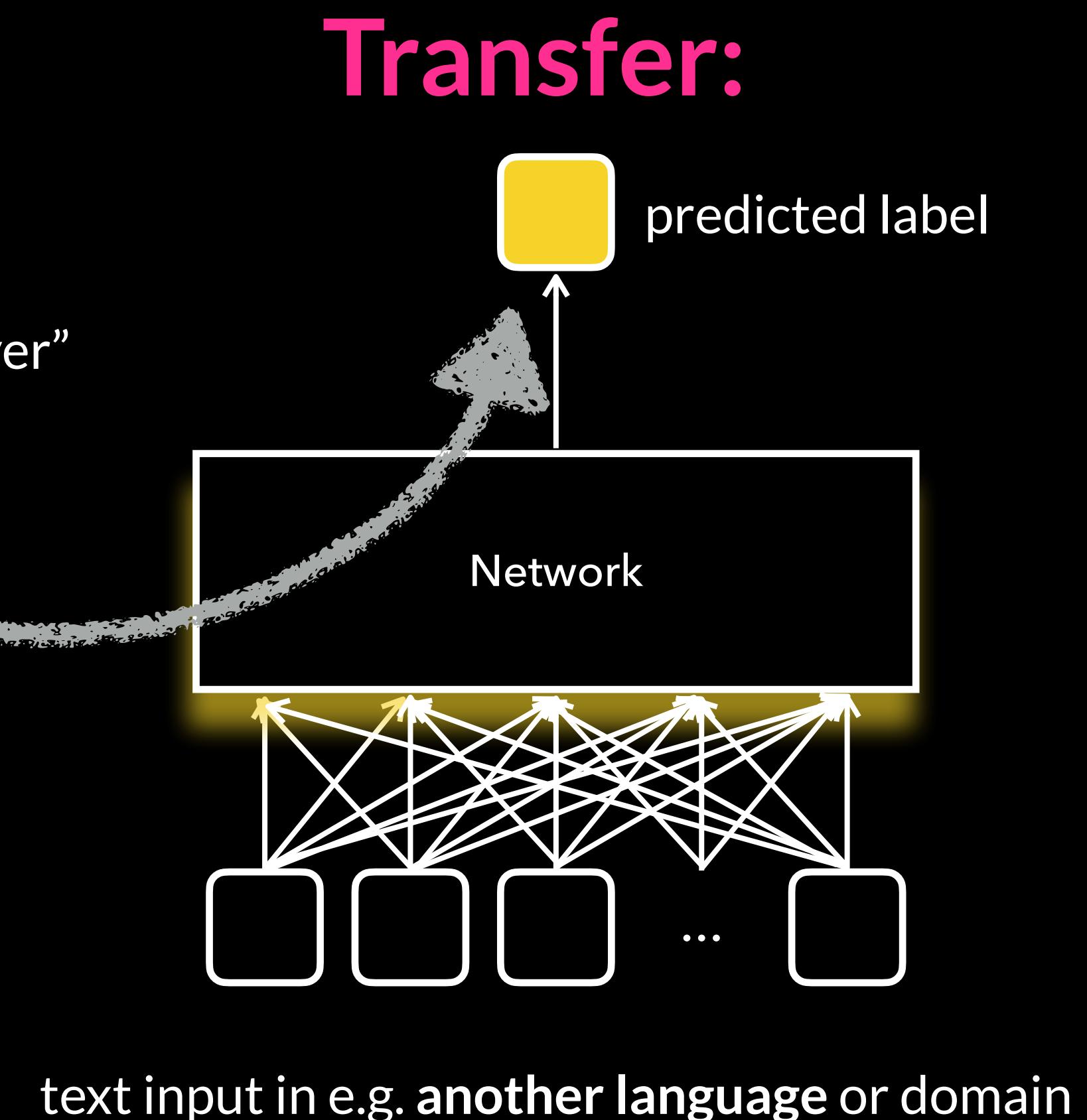
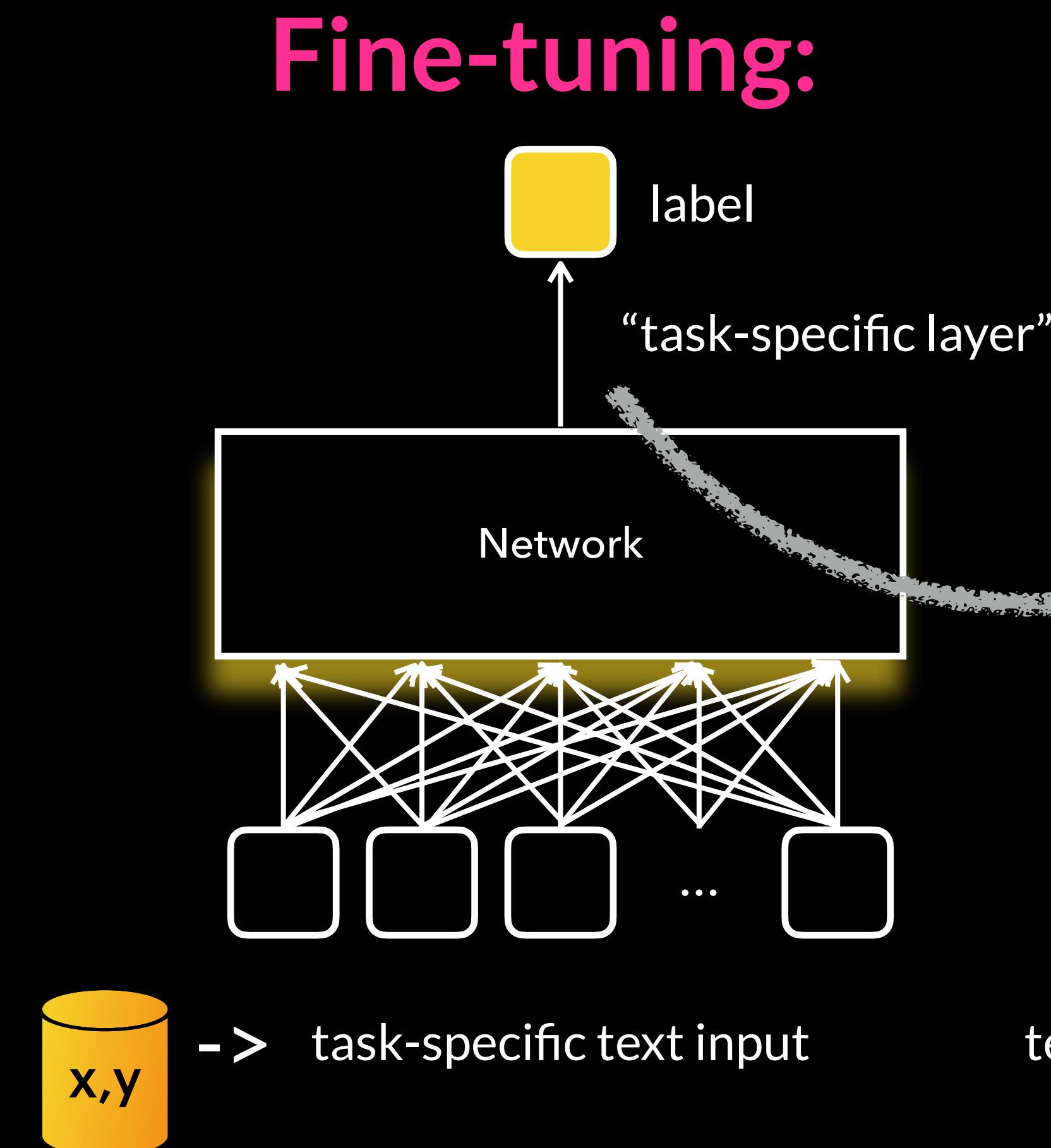
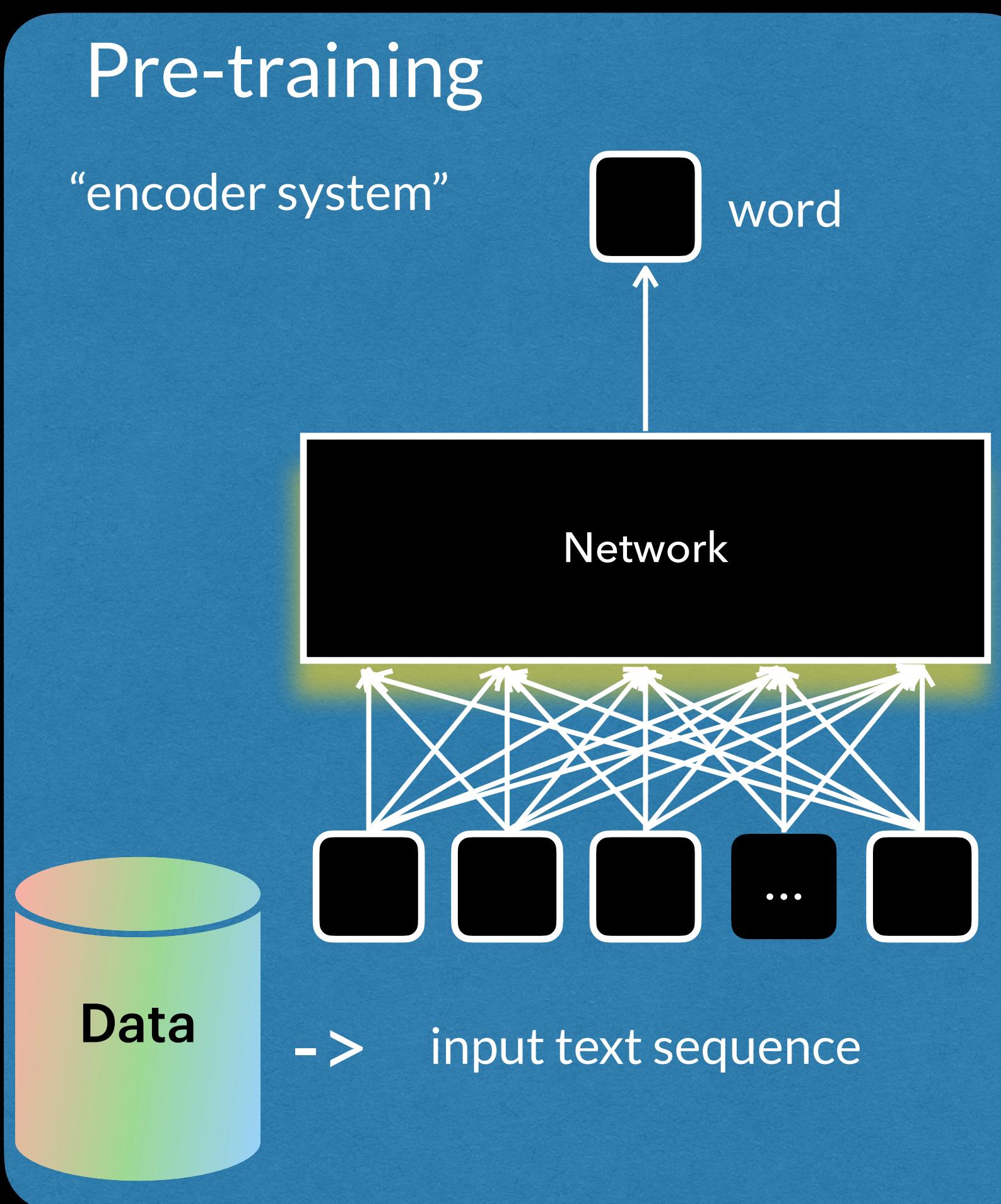
- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



# **Pre-training & Fine-tuning**

# Pre-train / Fine-tune / Transfer Paradigm

- ▶ 2018-2021: LMs as text encoders (the quest of better word representations)

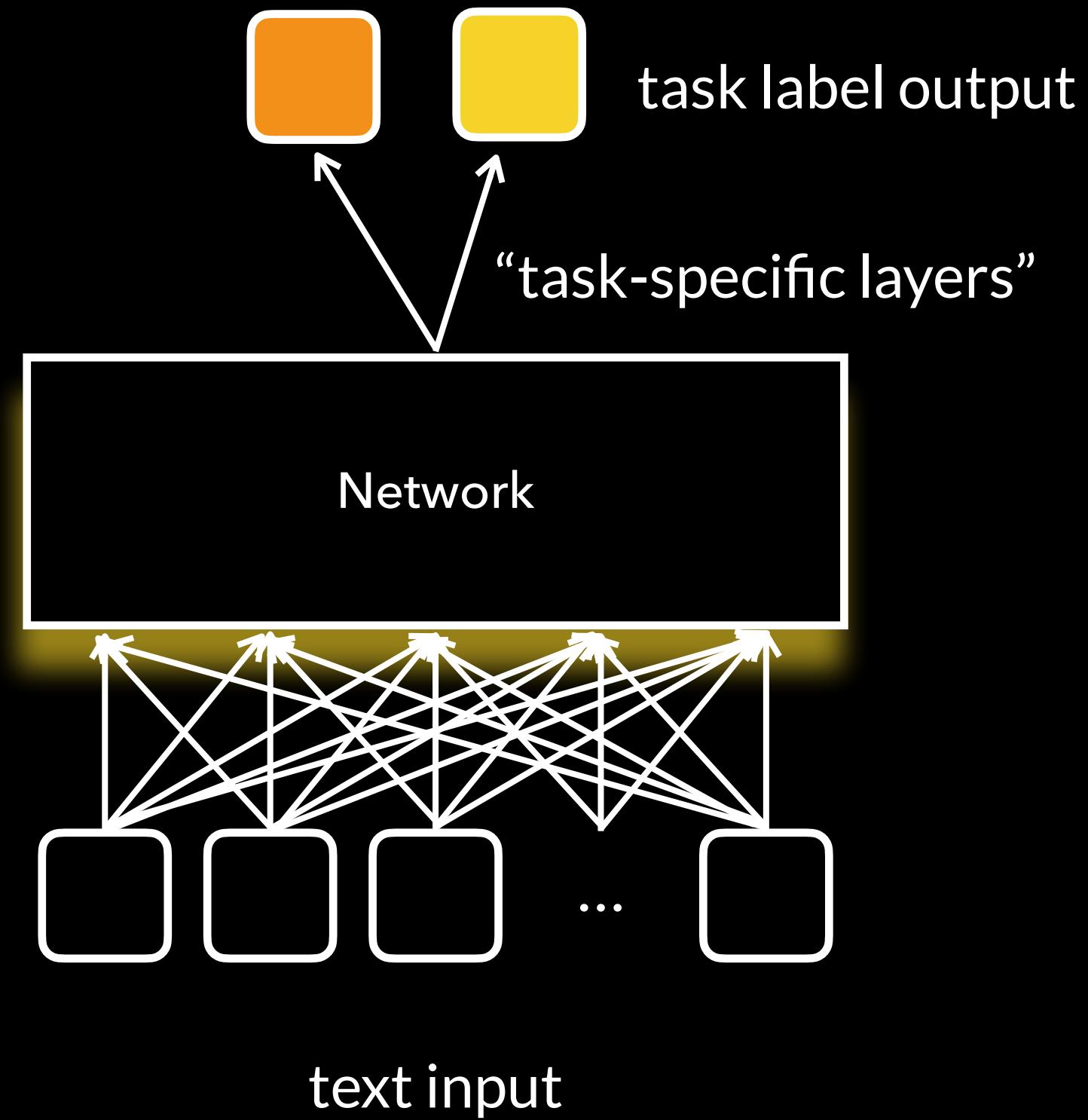


# In NLP we have a plethora of tasks

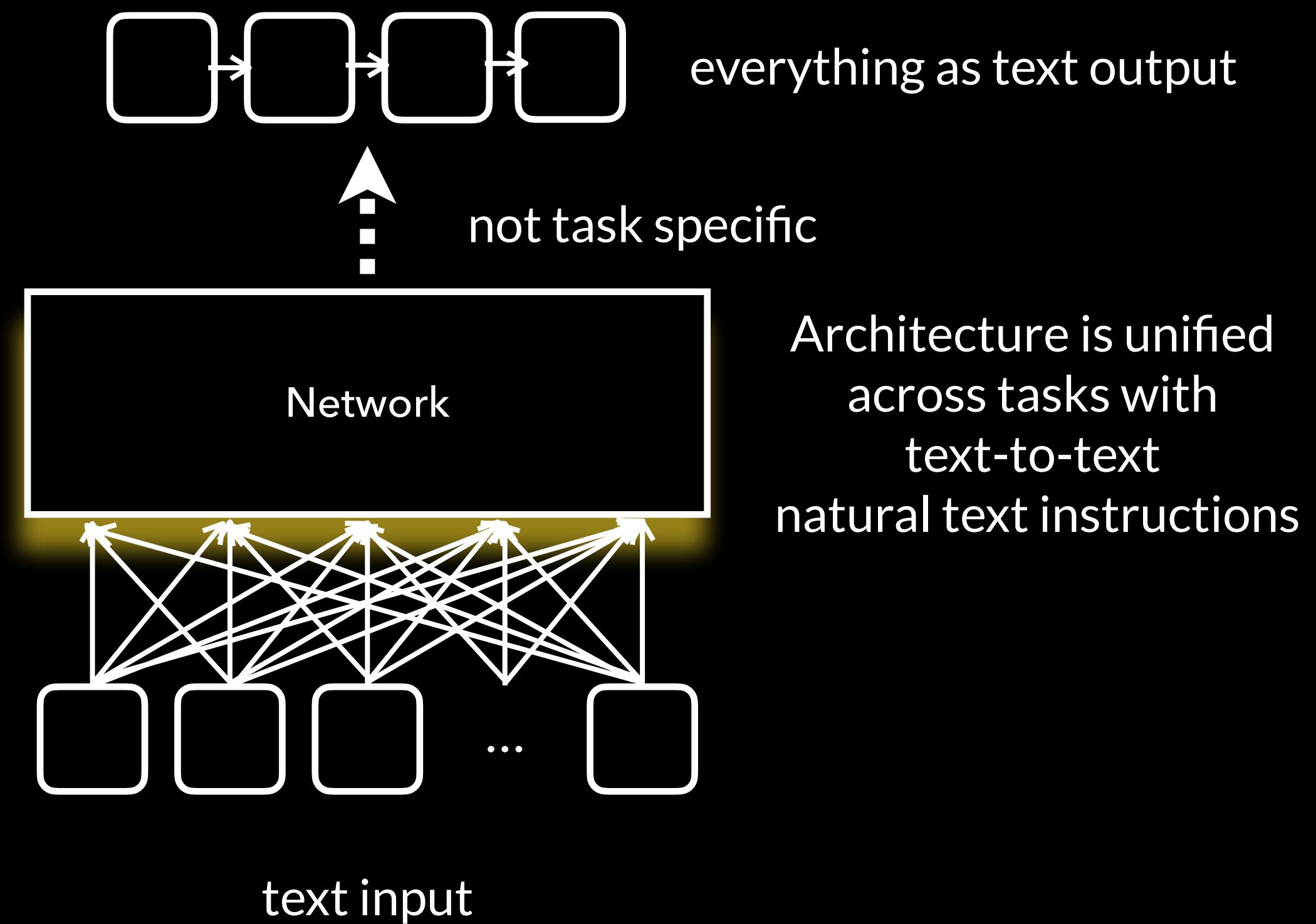
- ▶ Each NLP tasks requires different kinds of data:
  - ▶ **Language Modeling:** only text
  - ▶ **Machine translation:** naturally occurring parallel (translated) data
  - ▶ **Analysis tasks:** (from classification like sentiment analysis to structured prediction tasks like NER, semantic parsing or slot and intent detection:  
hand-labeled data)

# The shift to text-to-text transformers

## (Standard) Multi-task learning:



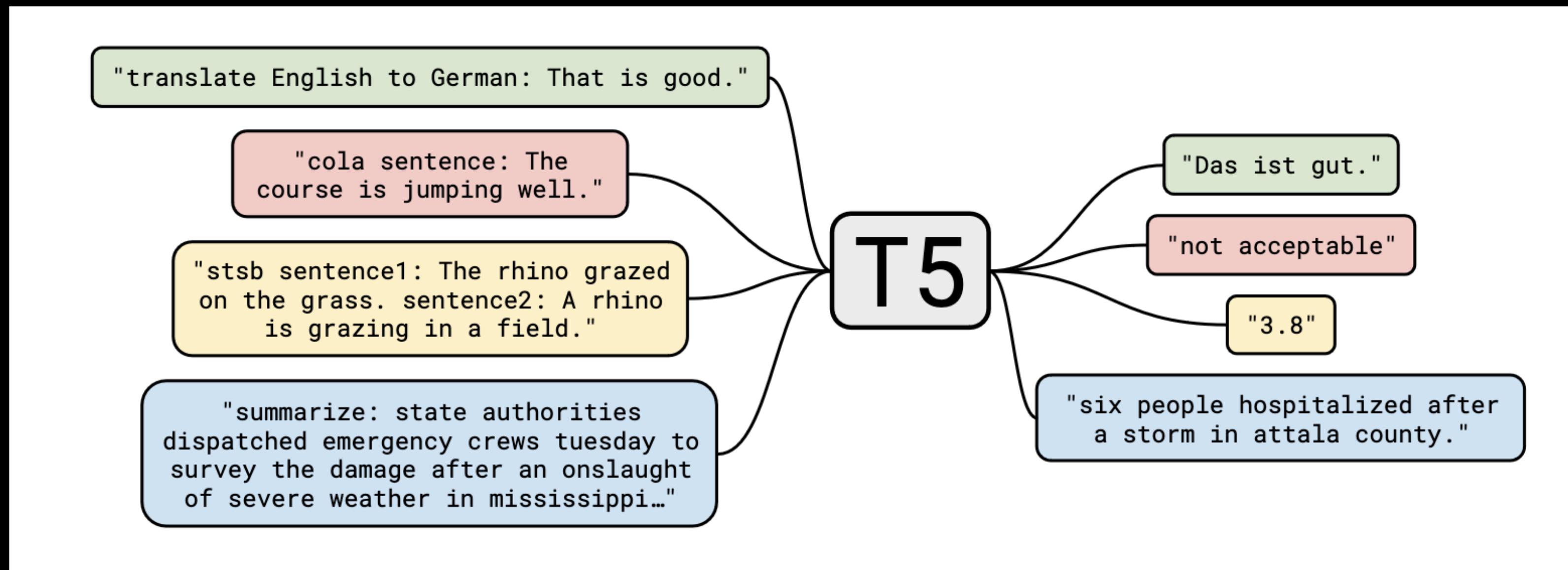
## Text-to-text format:



- ▶ 2019: from task-specific training/architectures to uniform text-to-text formats
- ▶ General knowledge (pre-training) + instruction fine-tuning => **Generalisation to new tasks**

# Instruction Fine-tuned LMs

- ▶ Learn many tasks in a single system (e.g. T5 Raffel et al., 2019): examples with instructions
- ▶ Massive Multi-Task learning (in text-to-text format)
- ▶ = General knowledge (pre-training) + ability to follow instructions (instruction fine-tuning)



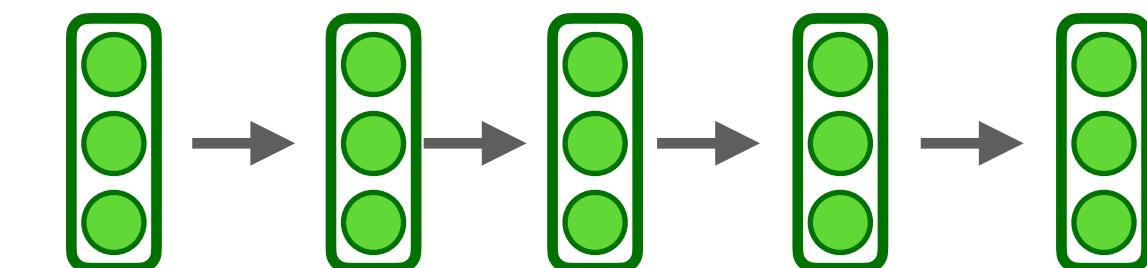
# T5 / encoder-decoder models

## Takeaways

- T5 (and BART) are very useful for all sorts of sequence-to-sequence tasks with language
  - T5 comes in different sizes
  - There are various customization (e.g., CodeT5)
- Extended the generalizations conclusions from BERT, and demonstrated the impact of data scale

# Today's roadmap

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ **Prompting, LLMs & Caution**



# Prompting

We thank the following sources for the materials on which slides on this section are based: Greg Durrett & Yoav Arzi

# A way to control LLMs: Prompting

- ▶ LLMs offer a completely new mode of operation that **does not require any change to their parameters**: prompting
- ▶ Prompt = Instruction given to the LLM
- ▶ Adapting the instructions to get different responses:
  - ▶ change the prompt (style)



Give me ideas for a tandem talk|

Sure, here are some engaging tandem talk ideas suitable for various contexts such as conferences, educational settings, or informal gatherings:

Give me ideas for a tandem talk. Please answer like  
writing to the queen

Your Majesty,

It is with the greatest respect and honor that I present to you several refined and engaging ideas for a tandem talk, worthy of the highest echelons of discourse. These topics, carefully chosen for their relevance and profundity, are designed to illuminate and inspire, reflecting the grandeur of your esteemed gatherings.

# Types of Prompting

- ▶ Zero-shot prompting (no examples)
- ▶ One-shot, Few-shot prompting (with one or few annotated examples)
- ▶ Chain-of-thought (CoT) Prompting and extensions thereof (asking for intermediate reasoning steps)

Please rate the following movie review from 1 to 5 (1=did not like it, 5=liked it very much).

Example:

Review: I was intrigued by the final scene despite thinking the opposite at the start.

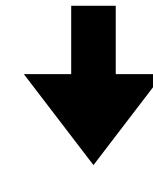
Rating: 4

Review: Despite the terrible soundtrack, Smith delivered a thoughtful agent.

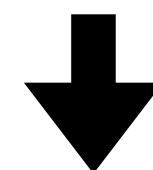
# Zero-shot Prompting

- Input: single unlabeled example  $\bar{x}$
- Output: the label  $\bar{y}$
- The task (and output) can be any text-to-text task: classification, summarization, translation
- Pre-processing: wrap  $\bar{x}$  with a template using a **verbalizer**  $v$
- The template controls the output

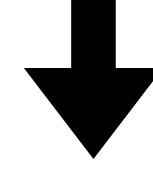
$\bar{x} = \text{the movie's acting could've been better, but the visuals and directing were top-notch.}$



$v(\bar{x}) = \text{Review: the movie's acting could've been better, but the visuals and directing were top-notch.}$   
**Out of positive, negative, or neutral this review is**



LLM

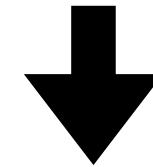


neutral  $\bar{y}$

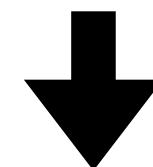
# In-context Learning (ICL)

- LLMs have the ability to “learn” to complete tasks through training in the prompt
- The recipe is simple:
  - Take a small number of annotated training example  $\{(\bar{x}^{(i)}, \bar{y}^{(i)})\}_{i=1}^N$
  - Convert them using verbalizer  $v$  templates
  - Concatenate them and follow with the target input  $\bar{x}$
  - The completion will be the label of the input

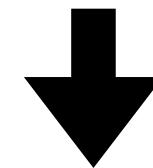
$\bar{x}$  = the movie's acting could've been better, but the visuals and directing were top-notch.



Review: The cinematography was stellar; great movie!  
Sentiment (positive or negative): positive  
Review: The plot was boring and the visuals were subpar.  
Sentiment (positive or negative): negative  
Review: The movie's acting could've been better, but the visuals and directing were top-notch.  
Sentiment (positive or negative):



LLM

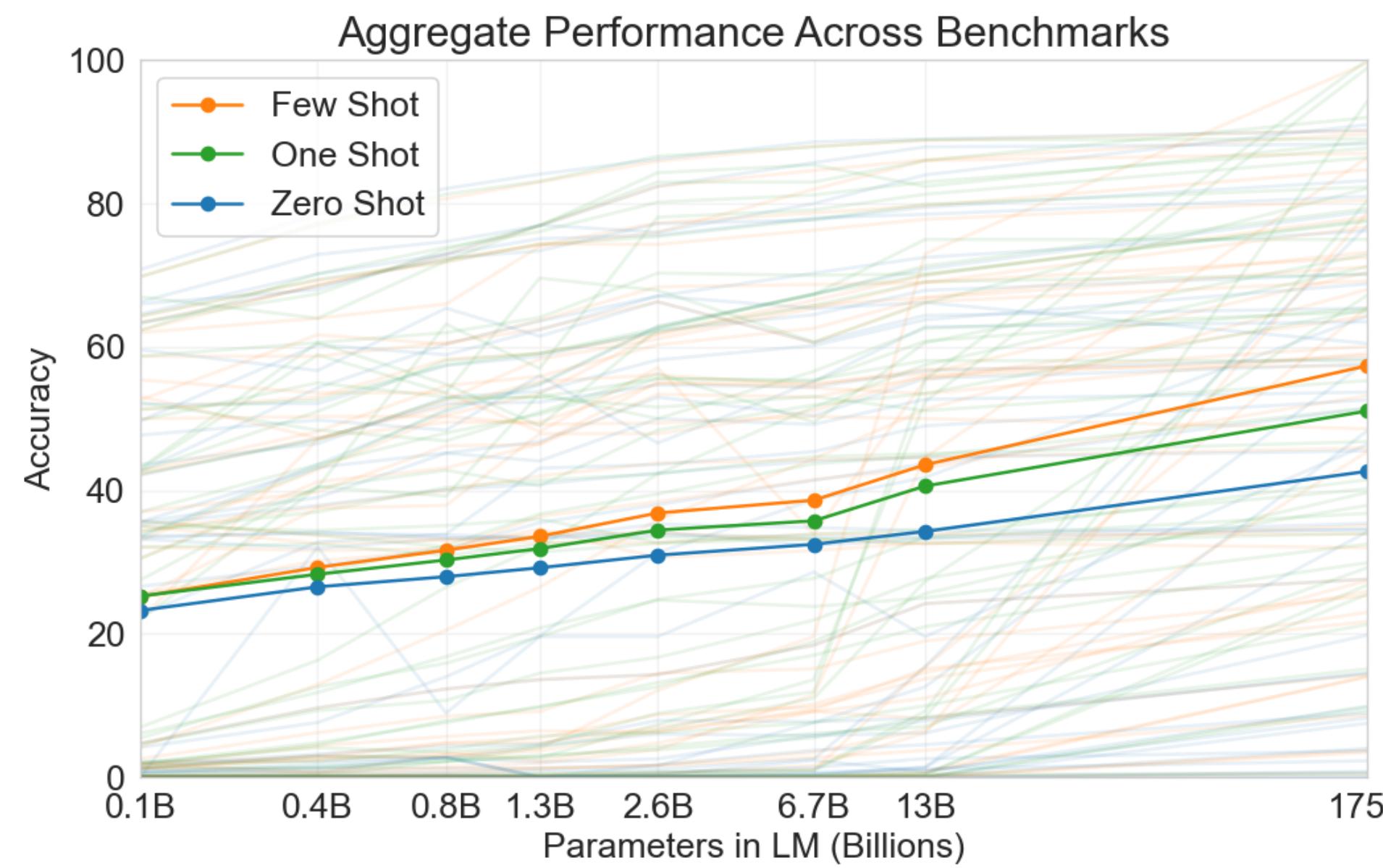


positive  $\bar{y}$

# In-context Learning (ICL)

## Performance

- Providing ICL examples almost always leads to significant improvements



**Figure 1.3: Aggregate performance for all 42 accuracy-denominated benchmarks** While zero-shot performance improves steadily with model size, few-shot performance increases more rapidly, demonstrating that larger models are more proficient at in-context learning. See Figure 3.8 for a more detailed analysis on SuperGLUE, a standard NLP benchmark suite.

# In-context Learning (ICL)

## Performance

- Providing ICL examples almost always leads to significant improvements
- Benefits tend to diminish with more examples

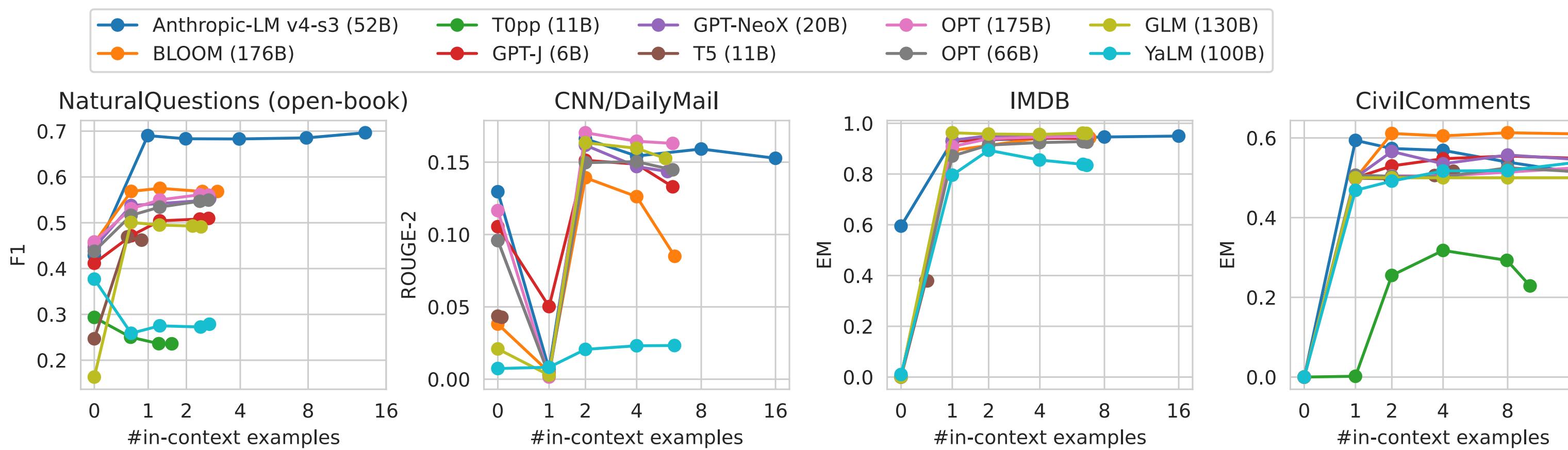


Figure 32: **Number of in-context examples.** For each model, we set the maximum number of in-context examples to [0, 1, 2, 4, 8, 16] and fit as many in-context examples as possible within the context window. We plot performance as a function of the average number of in-context examples actually used.

# Chain-of-Thought (CoT) Prompting - Example

- ▶ Adding “Let’s think step by step”

Sure, let's break this down step by step to come up with some compelling tandem talk ideas:

## Step 1: Identify the Audience and Purpose

Understanding who will be listening to the talk and what you aim to achieve is crucial. Is the audience academic, corporate, students, or a general public? Is the purpose to inform, entertain, persuade, or inspire?

## Step 2: Choose a Broad Topic

# CoT Prompting Example

► Standard Prompt

what is the size of the big lake closest to Verona

The largest lake closest to Verona is Lake Garda (Lago di Garda). Lake Garda is the largest lake in Italy, covering an area of approximately 370 square kilometers (143 square miles). It is situated about 30 kilometers (19 miles) west of Verona, making it easily accessible for a day trip or a short visit from the city.

► CoT (“Reasoning steps”):

what is the size of the big lake closest to Verona. Let's think step by step

To determine the size of the largest lake closest to Verona, we need to follow these steps:

1. Identify the major lakes near Verona:
  - Lake Garda
  - Lake Iseo
  - Lake Como
2. Determine the distances from Verona to these lakes:
  - Lake Garda: About 30 km west of Verona
  - Lake Iseo: About 90 km northwest of Verona
  - Lake Como: About 130 km northwest of Verona
3. Compare the sizes of these lakes:
  - Lake Garda: 370 square kilometers
  - Lake Iseo: 65.3 square kilometers
  - Lake Como: 146 square kilometers

# CoT Prompting Example 2

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. 

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

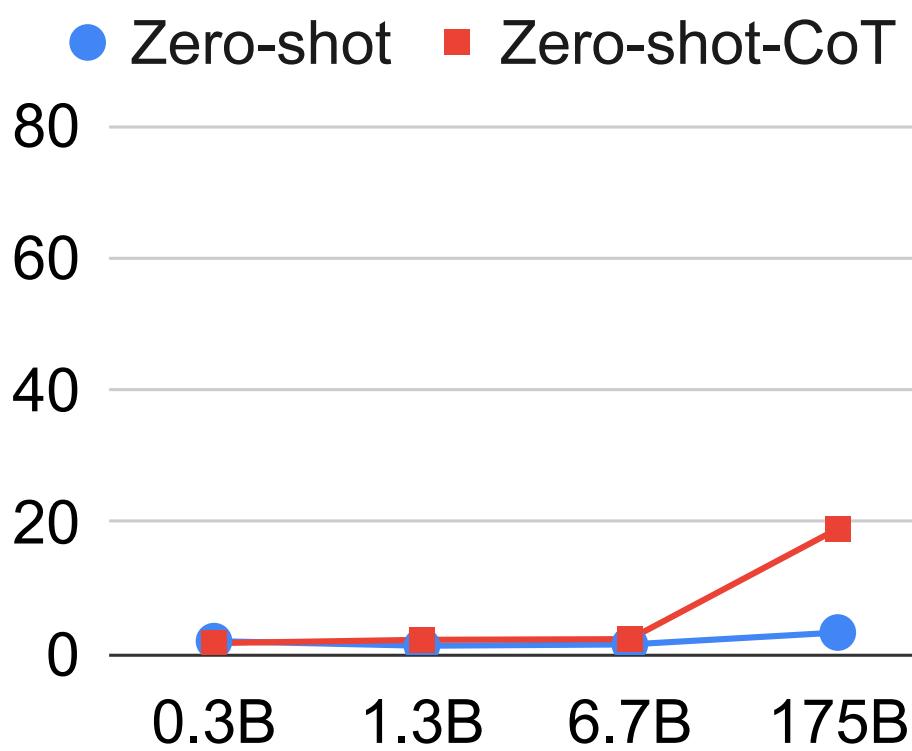
### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. 

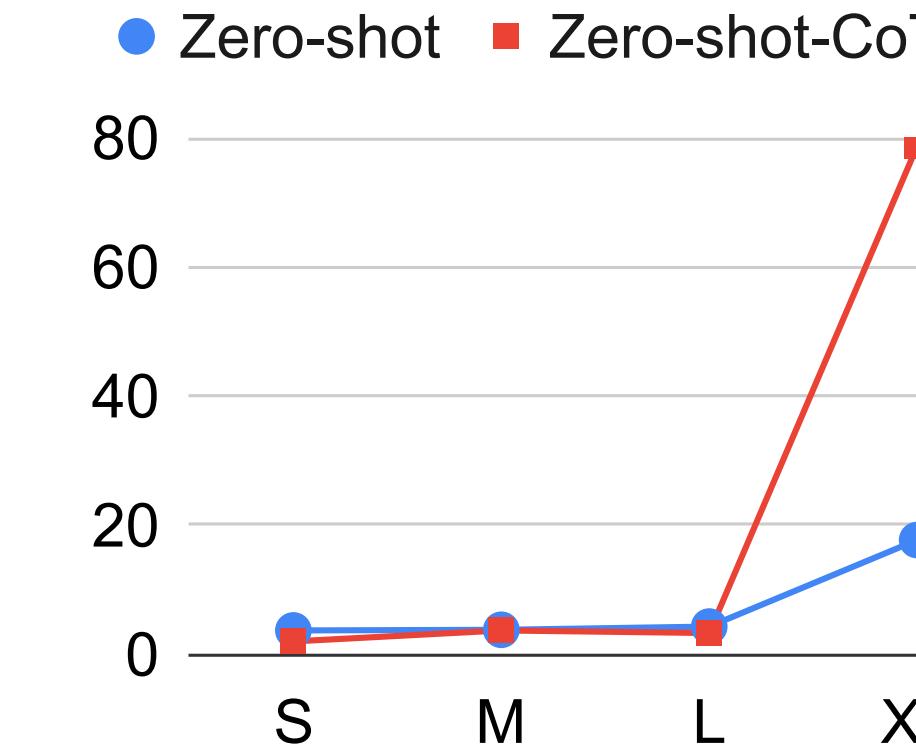
# Chain-of-thought (CoT) Prompting

## Step-by-step

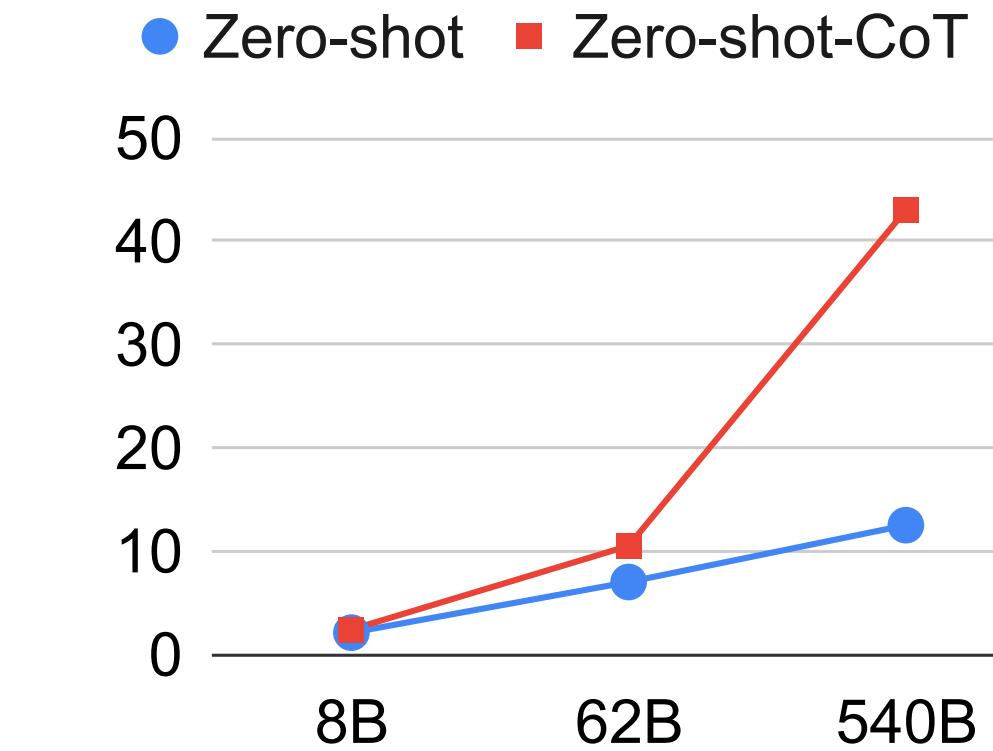
- Main idea: just “tell” the model to reason in steps
- Can significantly outperform zero-shot prompting with very large models



(a) MultiArith on Original GPT-3



(b) MultiArith on Instruct GPT-3



(c) GMS8K on PaLM

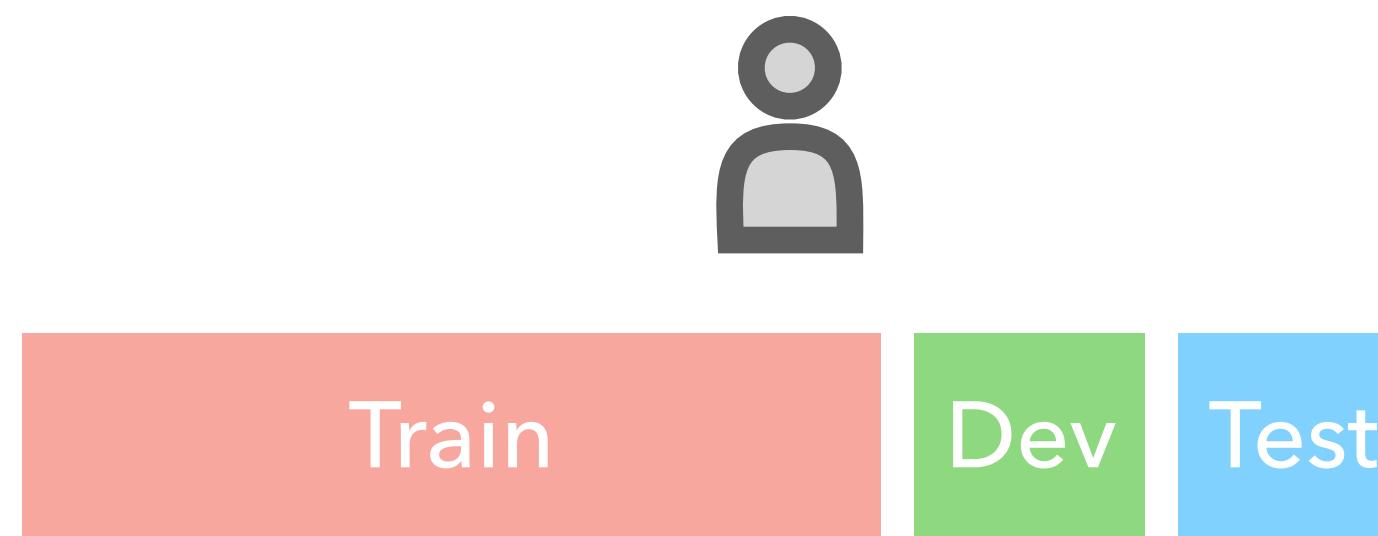
# Prompting LLMs - caution!

---

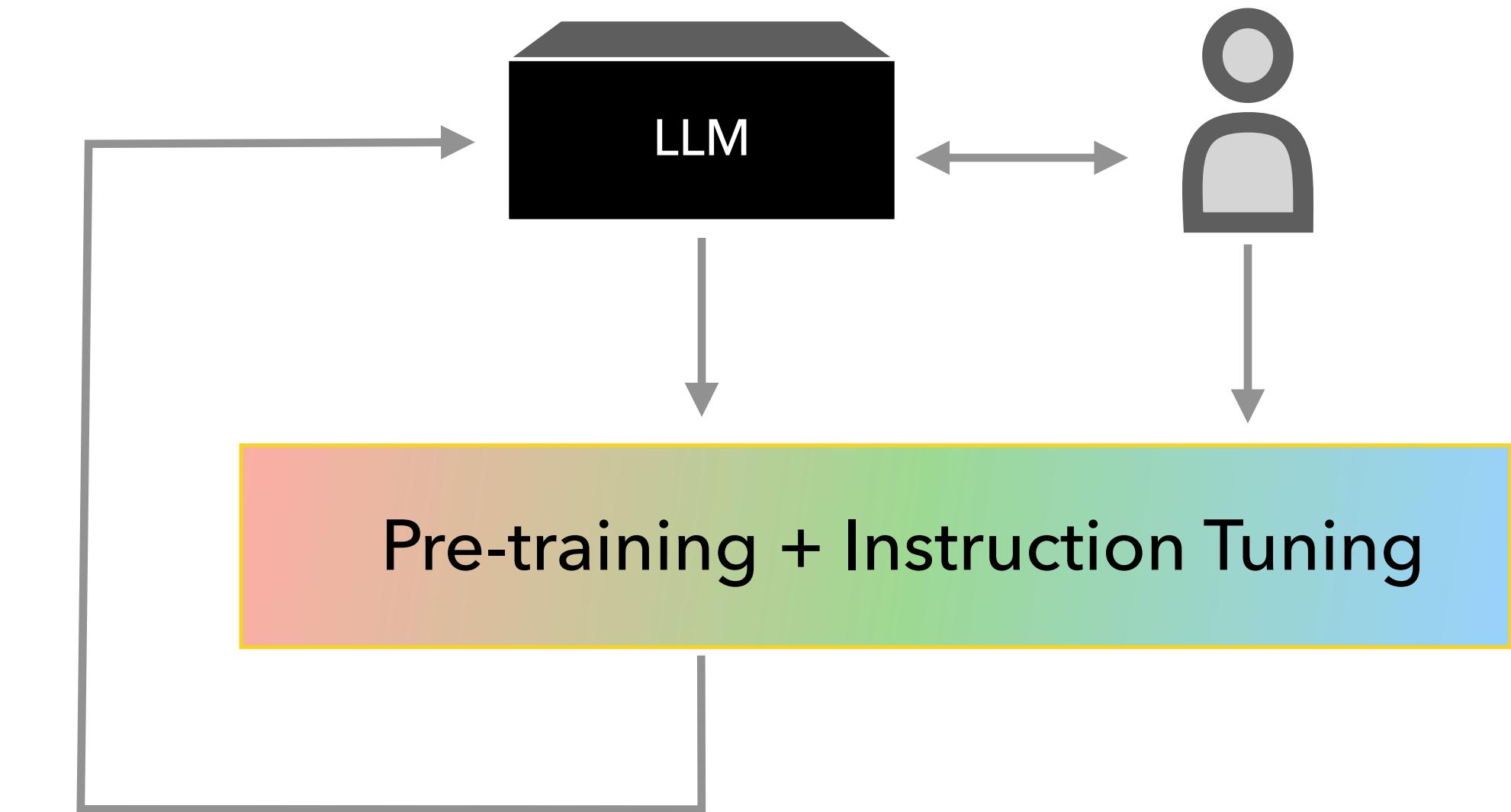
# Model Behaviour: Trustworthiness in the Era of LLMs

- From a “compartmentalised” notion of language tasks in NLP
- To powerful but less interpretable black-box models
- What is a “language task” today?

Compartmentalised NLP



Current Trend (in NLP)



# Model Behaviour: Does it Matter How we Prompt an LLM?

- ⚠ Performance is highly sensitive to the linguistic variation of a prompt

	prop.	prompt
mood	inter.	Do you find this movie review positive?
	indic.	You find this movie review positive.
	imper.	Tell me if you find this movie review positive.
aspt.	active	Do you find this movie review positive?
	pass.	Is this movie review found positive?
tense	past	Did you find this movie review positive?
	pres.	Do you find this movie review positive?
	future	Will you find this movie review positive?
modality	can	Can you find this movie review positive?
	could	Could you find this movie review positive?
	may	May you find this movie review positive?
	might	Might you find this movie review positive?
	must	Must you find this movie review positive?
	should	Should you find this movie review positive?
	would	Would you find this movie review positive?
synonymy	aprai.	Do you find this movie appraisal positive?
	comm.	Do you find this movie commentary positive?
	criti.	Do you find this movie critique positive?
	eval.	Do you find this movie evaluation positive?
	review	Do you find this movie review positive?

**The language of prompting:  
What linguistic properties make a prompt successful?**

Leidinger, van Rooij, Shutova, EMNLP 2023 Findings.

Table 1: Examples of variation of linguistic properties

# Multiple-Choice Question Answering (MCQA) Prompt Style

---

**General Instruction:** Please read the multiple-choice question below carefully and select ONE of the listed options and only give a single letter.

**Question:** The Web was effectively invented by Berners-Lee in which year?

**Options:**

- A. 1991
- B. 1980
- C. 1989
- D. 1993

**Answer:**



# Evaluation Protocols: Do Answer Options Impact LLM Outputs?

- ! LLM's "A"-bias in MCQA responses

Choice ordering 1

**Question:** In the past 12 months, has this person given birth to any children?

A. Yes

B. No

**Answer:**

Response

A. Yes B. No

P("A")	0.82	P("B")	0.11
--------	------	--------	------

Choice ordering 2

**Question:** In the past 12 months, has this person given birth to any children?

A. No

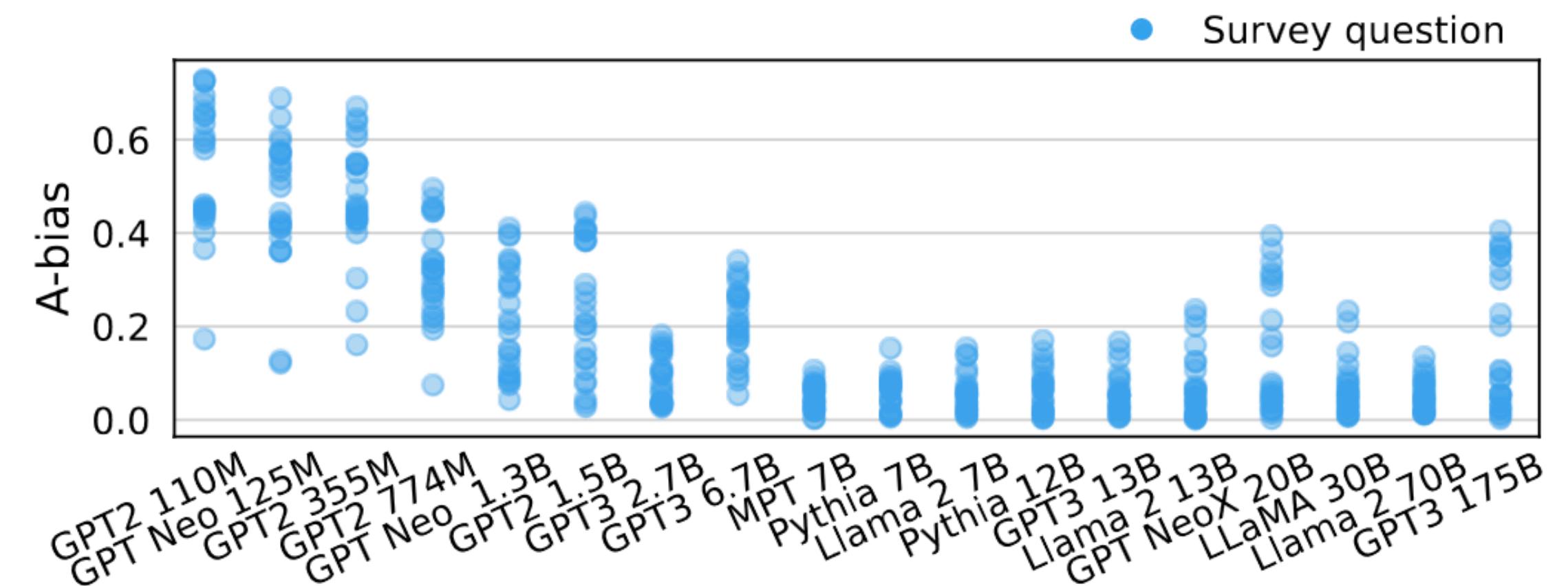
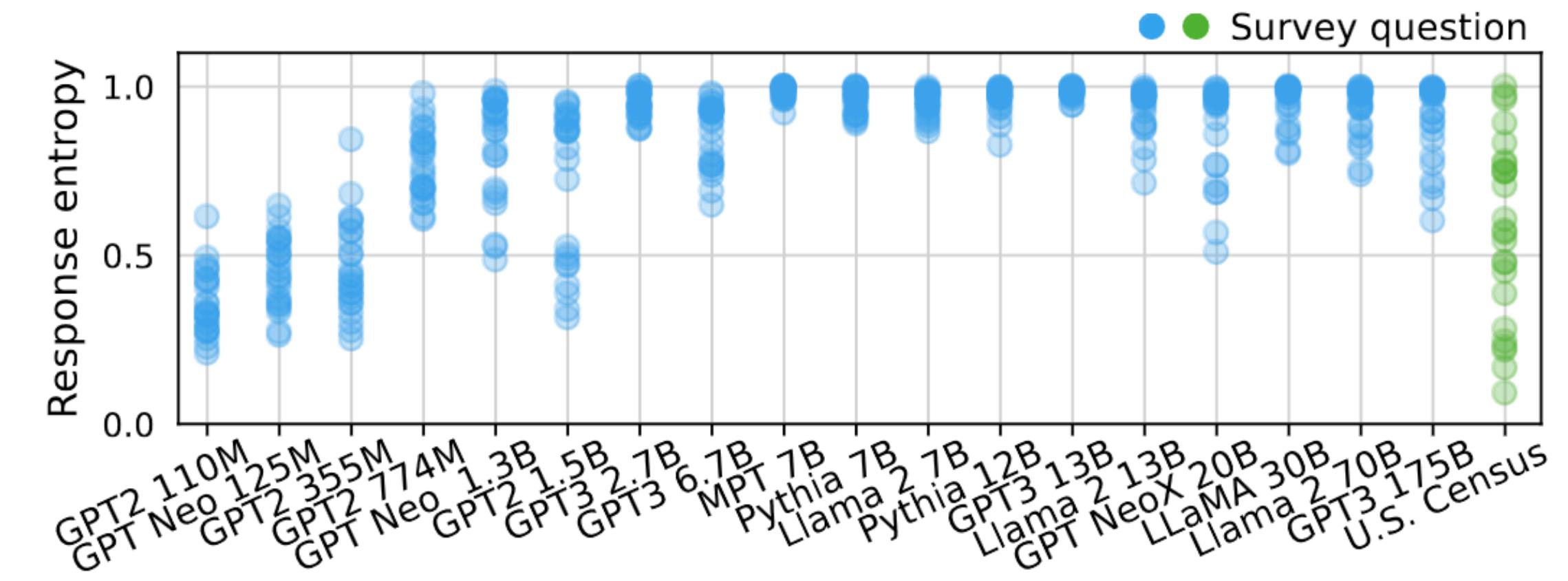
B. Yes

**Answer:**

Response

A. No B. Yes

P("A")	0.80	P("B")	0.15
--------	------	--------	------



# Evaluation Protocols: Does It Matter How We Extract Answers?

- ⚠️ But “First-token log probs” do not match the text answers

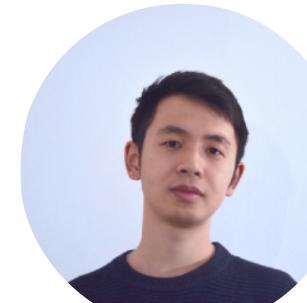
**General Instruction:** Please read the multiple-choice question below carefully and select ONE of the listed options and only give a single letter.

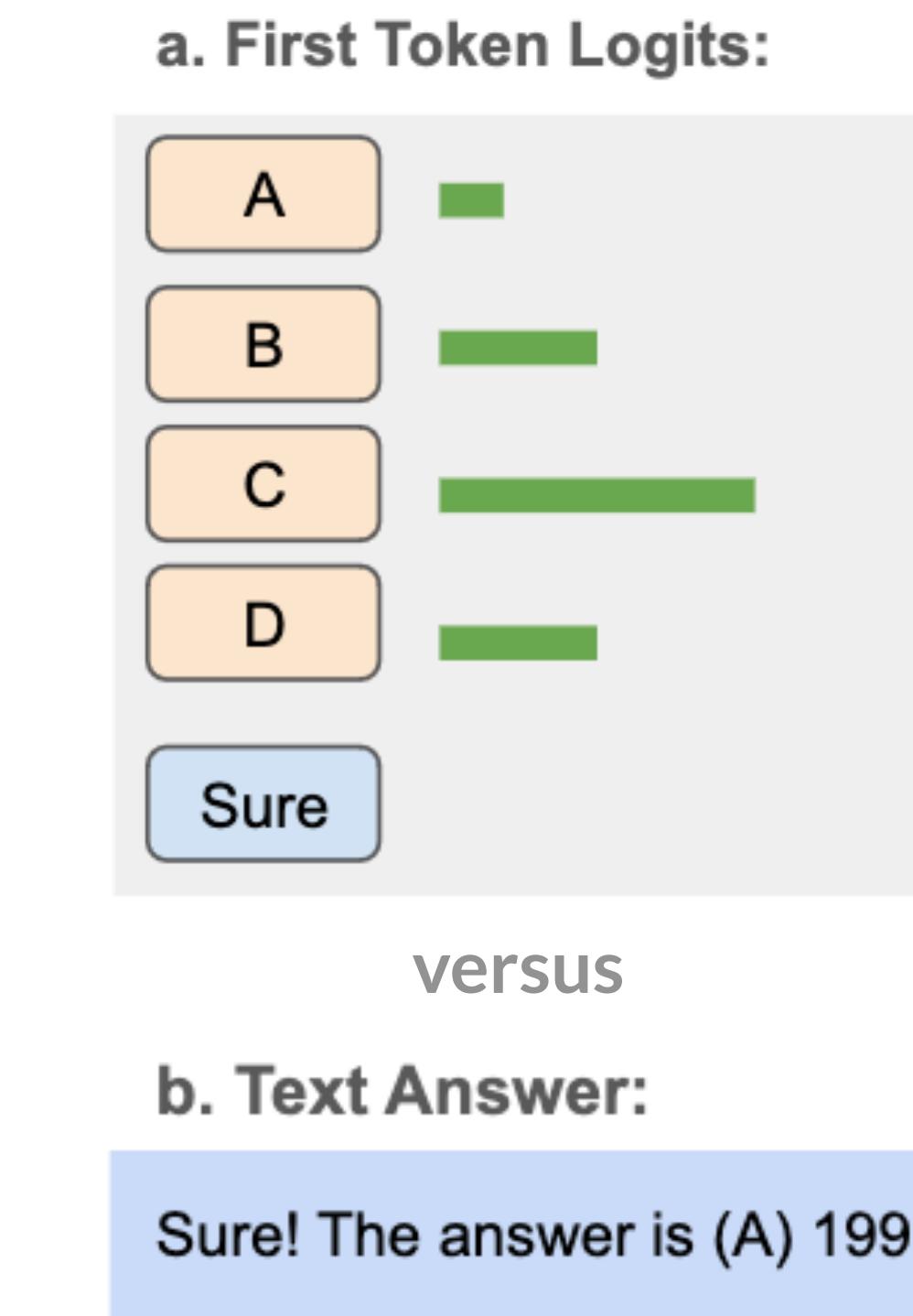
**Question:** The Web was effectively invented by Berners-Lee in which year?

**Options:**

- A. 1991
- B. 1980
- C. 1989
- D. 1993

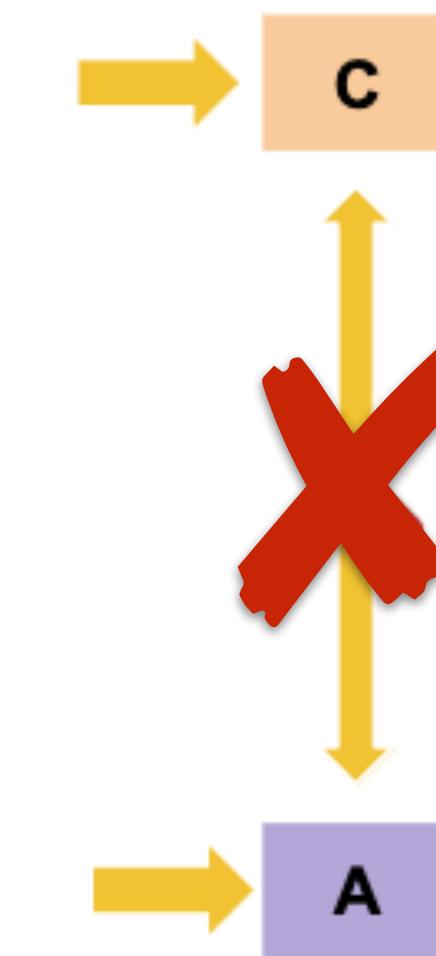
**Answer:**





Model (0-shot)	First Token	Text Answer
Gemma-7b-Inst	30.2	50.8
Llama2-7b-Chat	34.9	43.1
Llama2-13b-Chat	40.2	47.6
Mistral-7b-Inst-0.2	53.2	53.6

Performance on MMLU.



# Evaluation Protocols: Chain-of-Thought, Better and Sound?

## ► ! Chain-of-Thought Reasoning is Not Sound

Statements:

1. There is a purple marble in the box if and only if there is a blue marble in the box.
2. There is a blue marble in the box if and only if there is an olive marble in the box.
3. Either there is an olive marble in the box or else there is a red marble in the box, but not both.
4. Either there is a red marble in the box or else there is a green marble in the box, but not both.

Conclusion: If there is a purple marble in the box then there is a green marble in the box.

CoT response (excerpt) - manually verified:

7. Let's consider the possibility that there is an olive marble in the box. If that's the case, then, according to Statement 2, there must be a blue marble in the box as well.
8. Now, using Statement 1, we know that if there is a blue marble in the box, then there must be a purple marble in the box as well.
9. However, Statement 3 tells us that if there is an olive marble in the box, then there cannot be a red marble in the box. But, Statement 4 tells us that if there is a red marble in the box, then there cannot be a green marble in the box.
10. Since we cannot have both an olive marble and a red marble in the box, and we cannot have both a red marble and a green marble in the box, it seems that the only possibility left is that there is an olive marble in the box, and therefore, a blue marble, a purple marble, but no red or green marbles.
11. However, the conclusion states that "If there is a purple marble in the box then there is a green marble in the box", which is not logically true based on the given statements.  
Conclusion: False.



Supposition following (sound).

Invalid inference.



Conclusion (invalid).

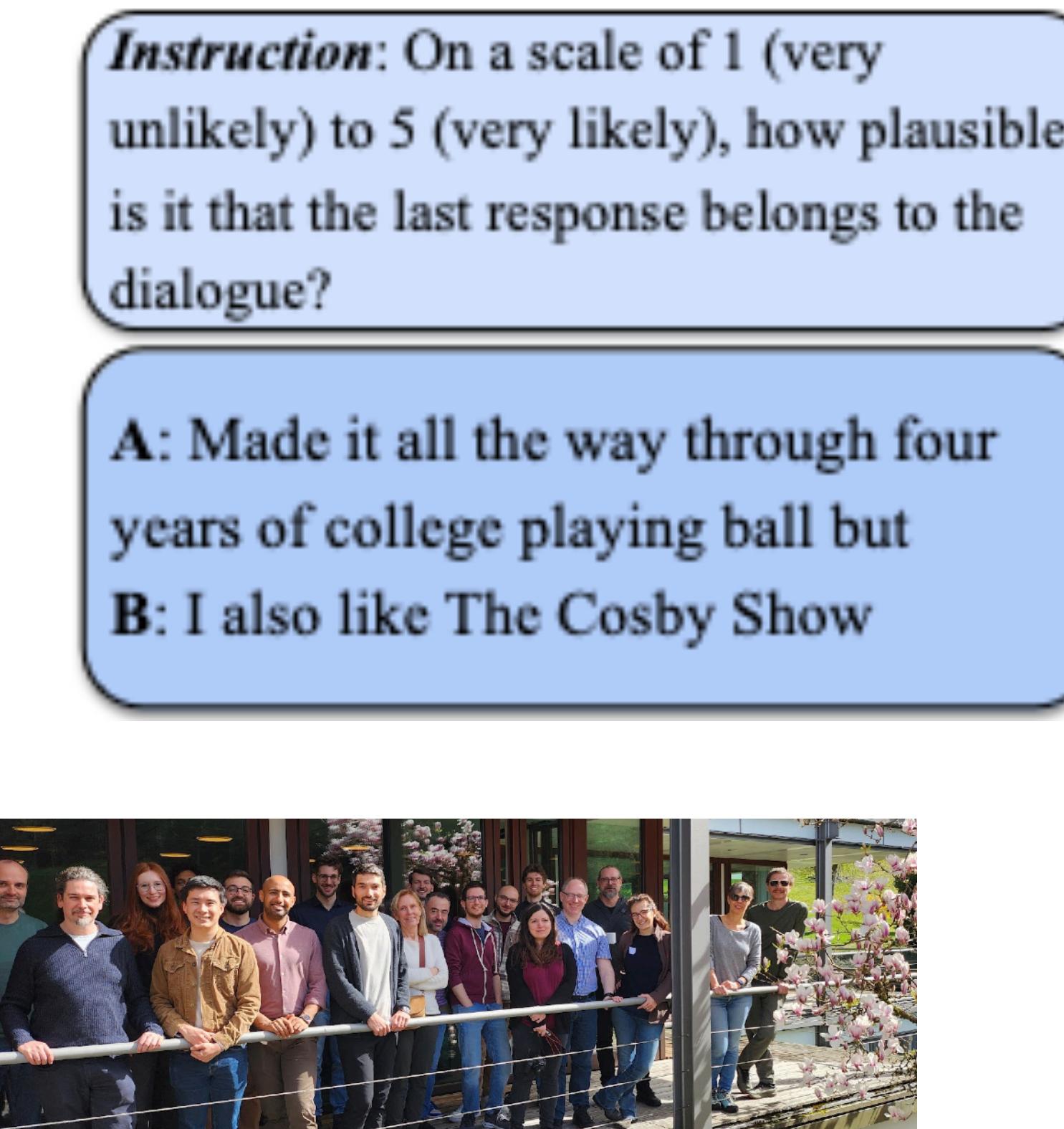
Final answer (incorrect).

Model	Correct Answer	Sound Reasoning
Zephyr-7B-β	45.0 ± 15.5	25.0 ± 10.5
Mistral-7B-Instruct	55.0 ± 10.0	25.0 ± 7.5
LLaMA-2-7B	46.7 ± 6.7	0.0 ± 0.0
LLaMA-2-13B	40.0 ± 8.2	15.0 ± 6.2
LLaMA-2-70B	56.7 ± 6.2	31.7 ± 9.7

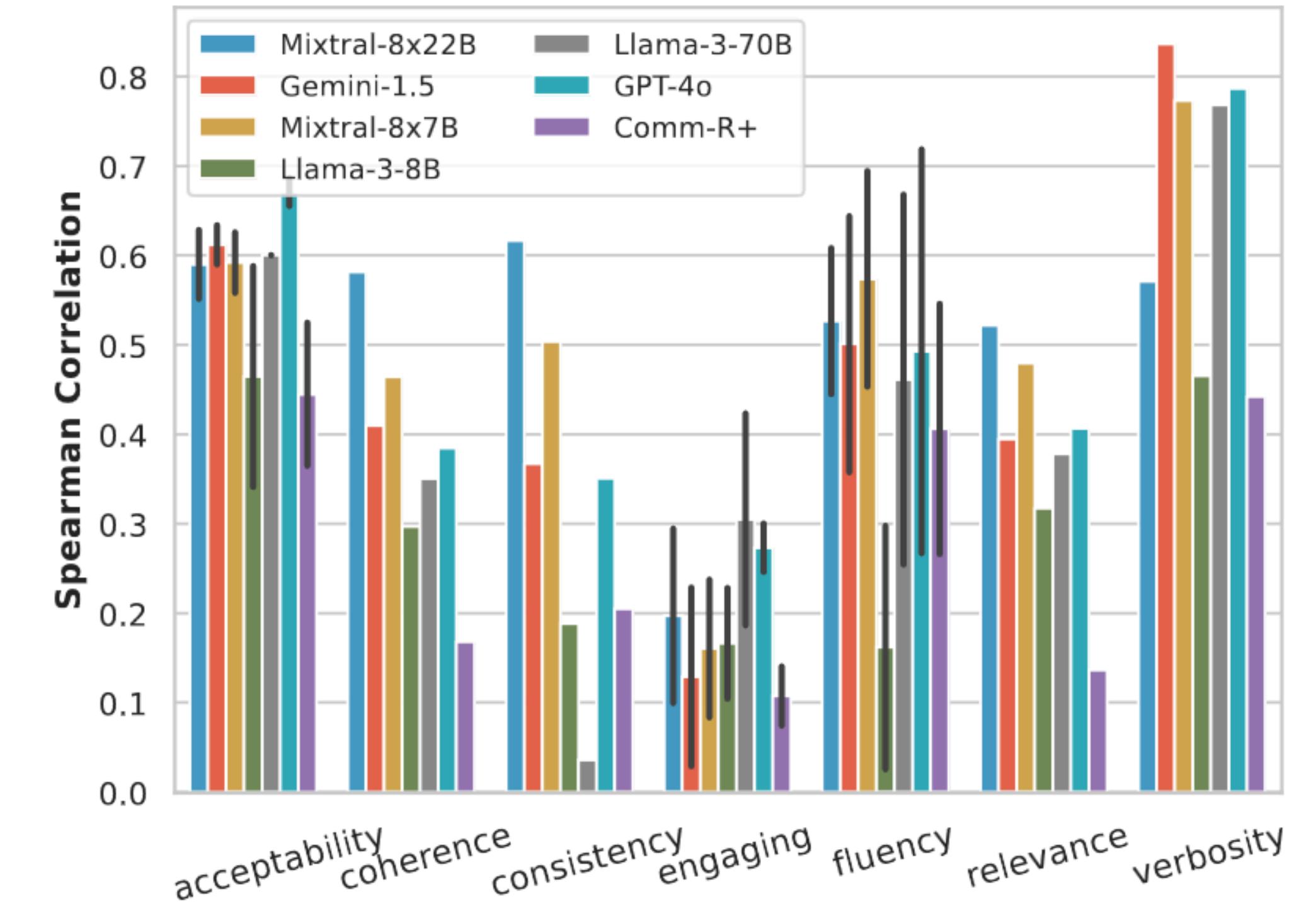
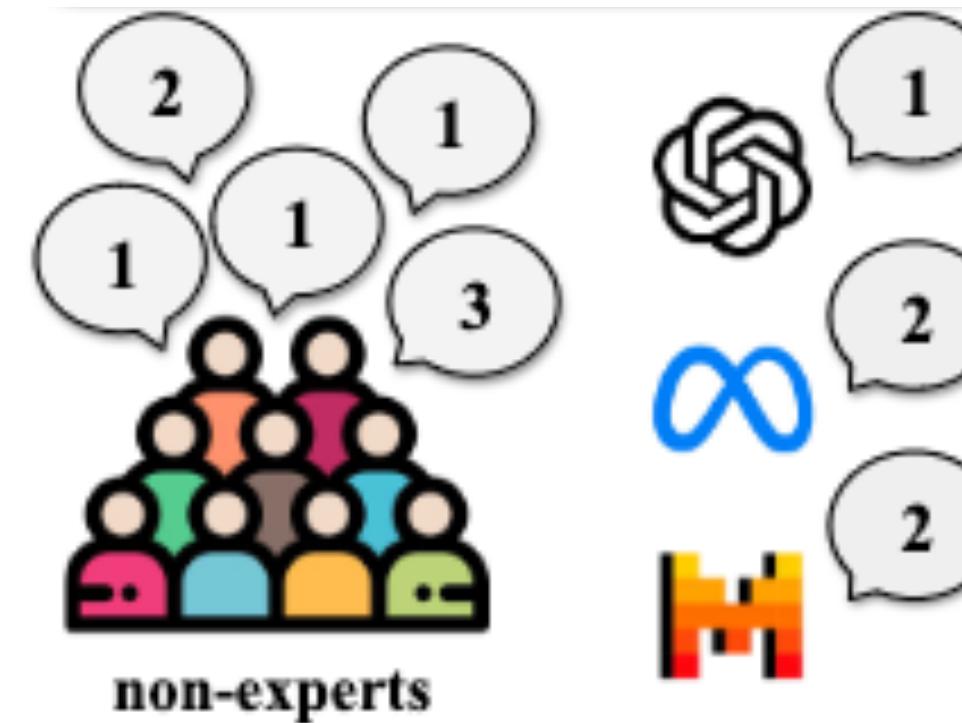
# Evaluation Protocols: Can LLMs Replace Humans Judges?

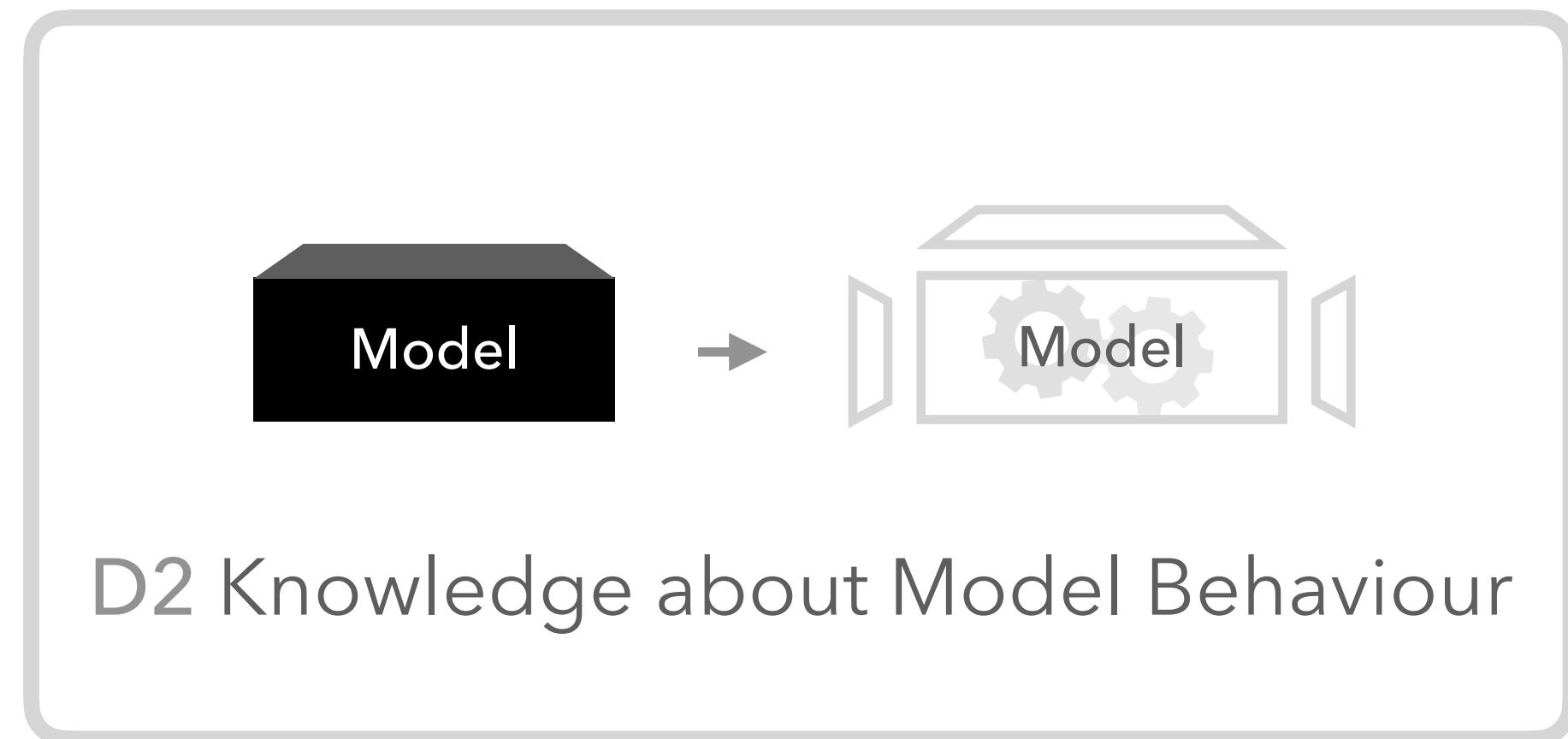
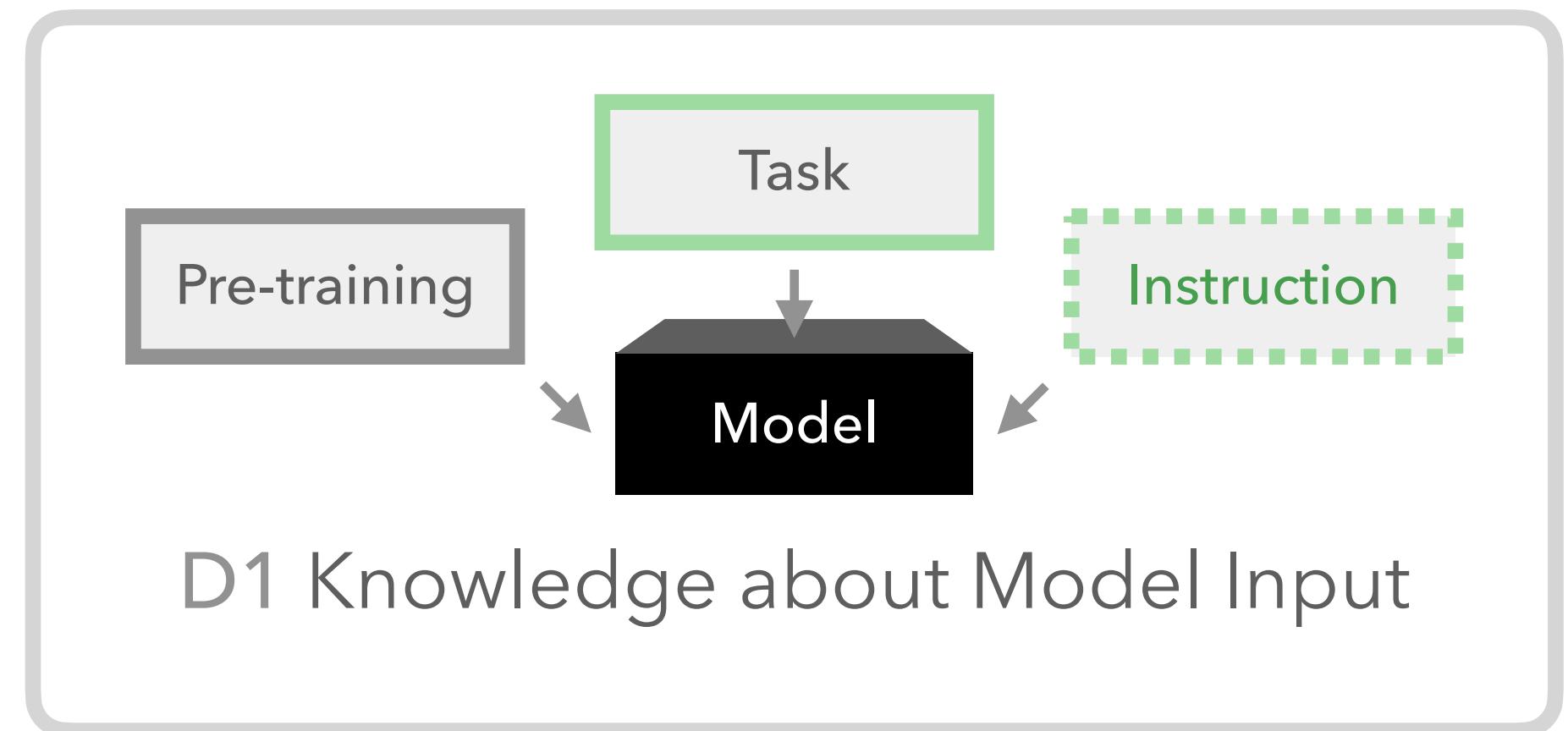
- ▶ ! A lot of variability in LLM outputs
- ▶ LLMs are not ready yet to replace human judges - not even GPT-4o:

E.g. Plausibility:



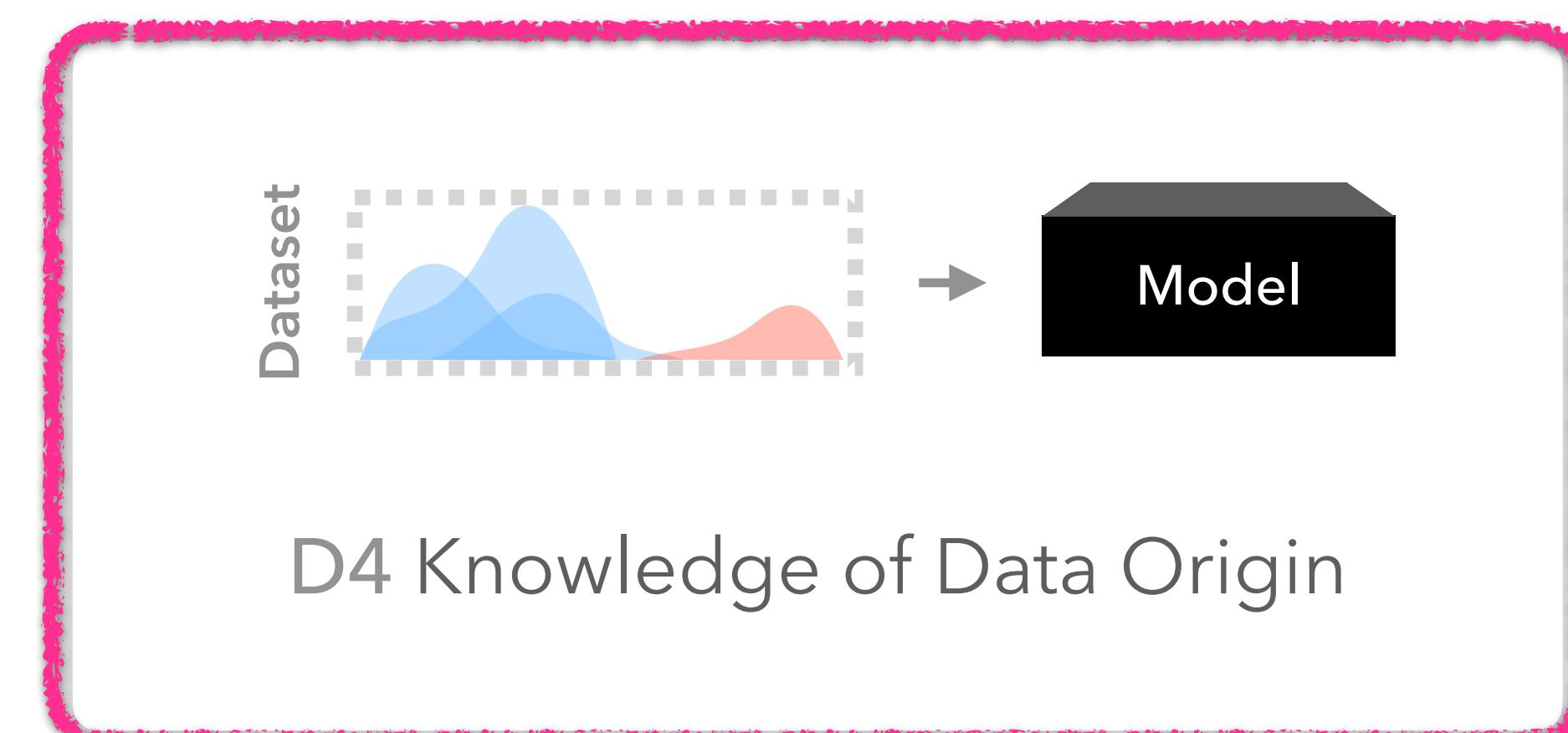
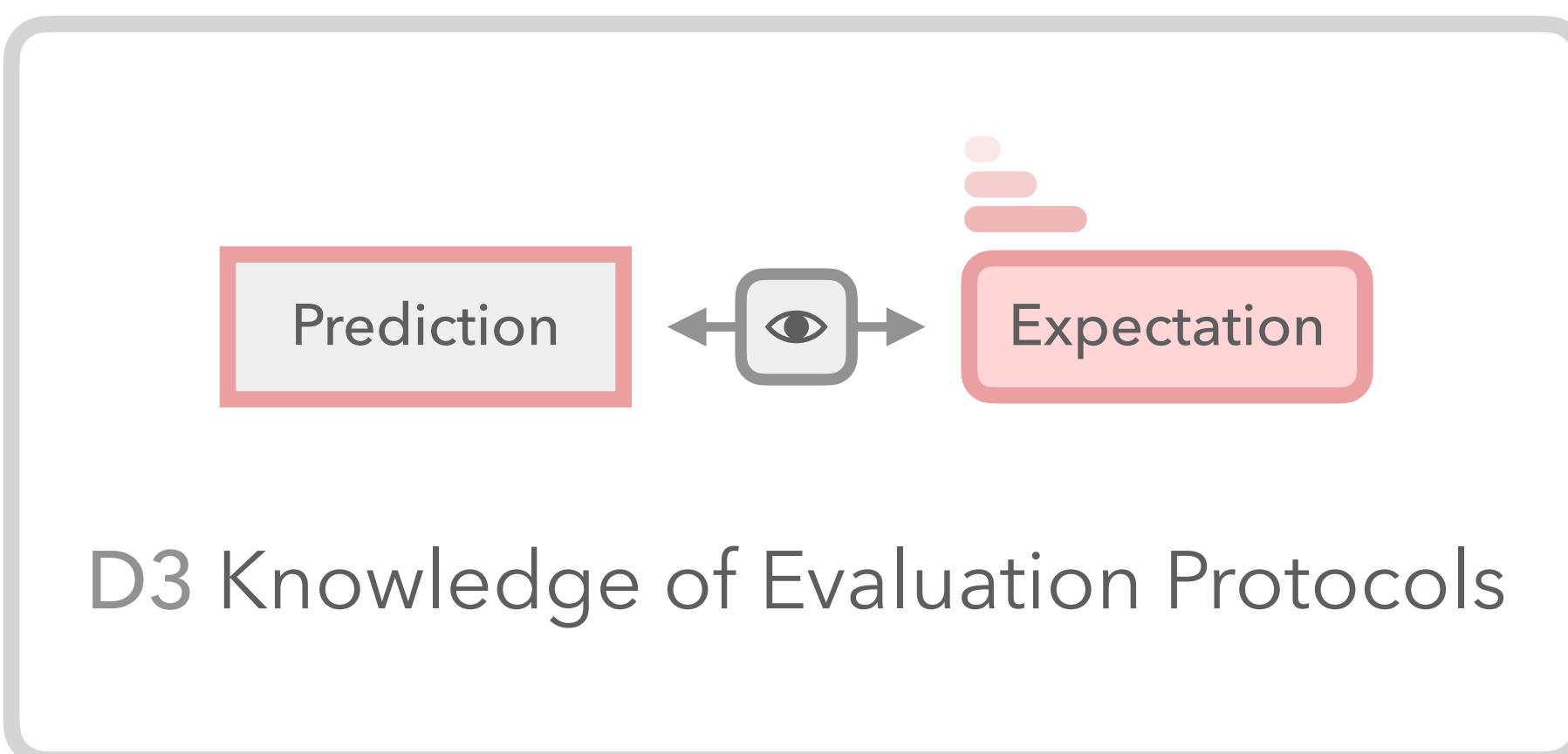
Humans vs Models:





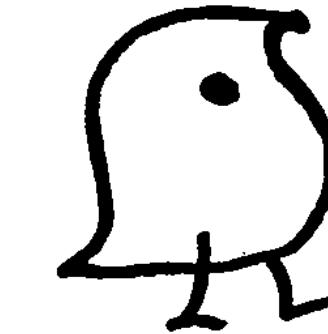
Trust arises from knowledge of origin as well as from knowledge of functional capacity.

*Trustworthiness - Working Definition by David G. Hays, 1979*

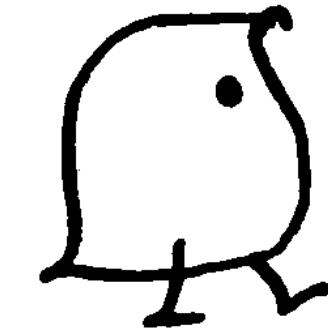


# Data Origin: Late 2022 Claim “ChatGPT Passes the Wug Test”

- ▶ ! No! Data contamination. Tested only on English.



THIS IS A WUG.



NOW THERE IS ANOTHER ONE.

THERE ARE TWO OF THEM.

THERE ARE TWO \_\_\_\_\_.



**David Mortensen** @dmort27 · Nov 1, 2023

To put it more simply, [@jamonholmgren's illustration is meaningless because GPT-3 can complete the prompt simply by recapitulating a conversation it has seen thousands of times in its training data.](#)

...



**Leonie Weissweiler** @LAWeissweiler · Oct 31, 2023

☀️ NEW PAPER ☀️

ChatGPT's language skills are impressive, but can it pass this simple test from 1958? In our #EMNLP2023 paper, @vjhofmann and I evaluate ChatGPT's ability for morphological generalization on a novel set of made-up "wug words" in four languages.



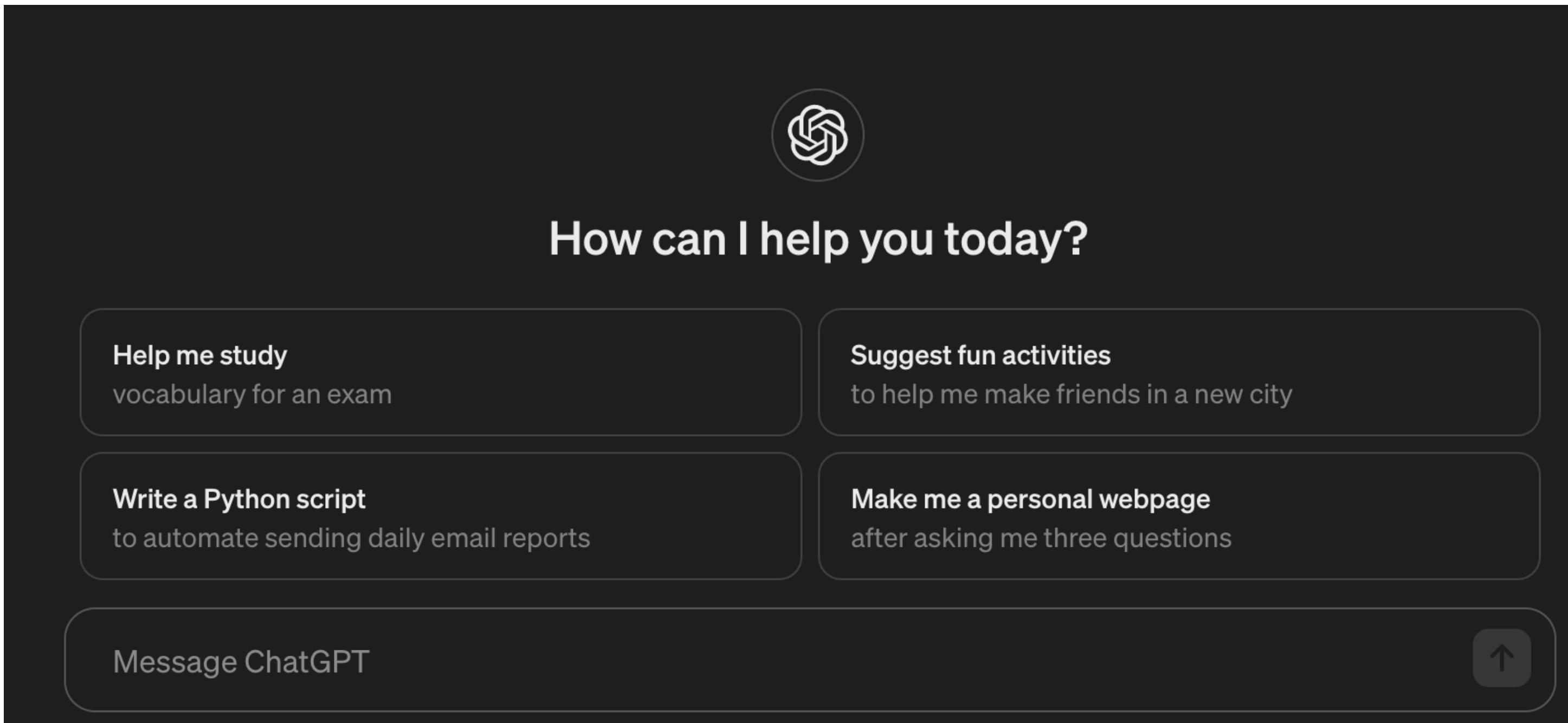
(1/4)

**Counting the Bugs in ChatGPT’s Wugs: A Multilingual Investigation into the Morphological Capabilities of a Large Language Model**

# Data Origin: Indirect Data Leakage

---

- ⚠ Data continuously provided by users (e.g. via OpenAI's web interface)



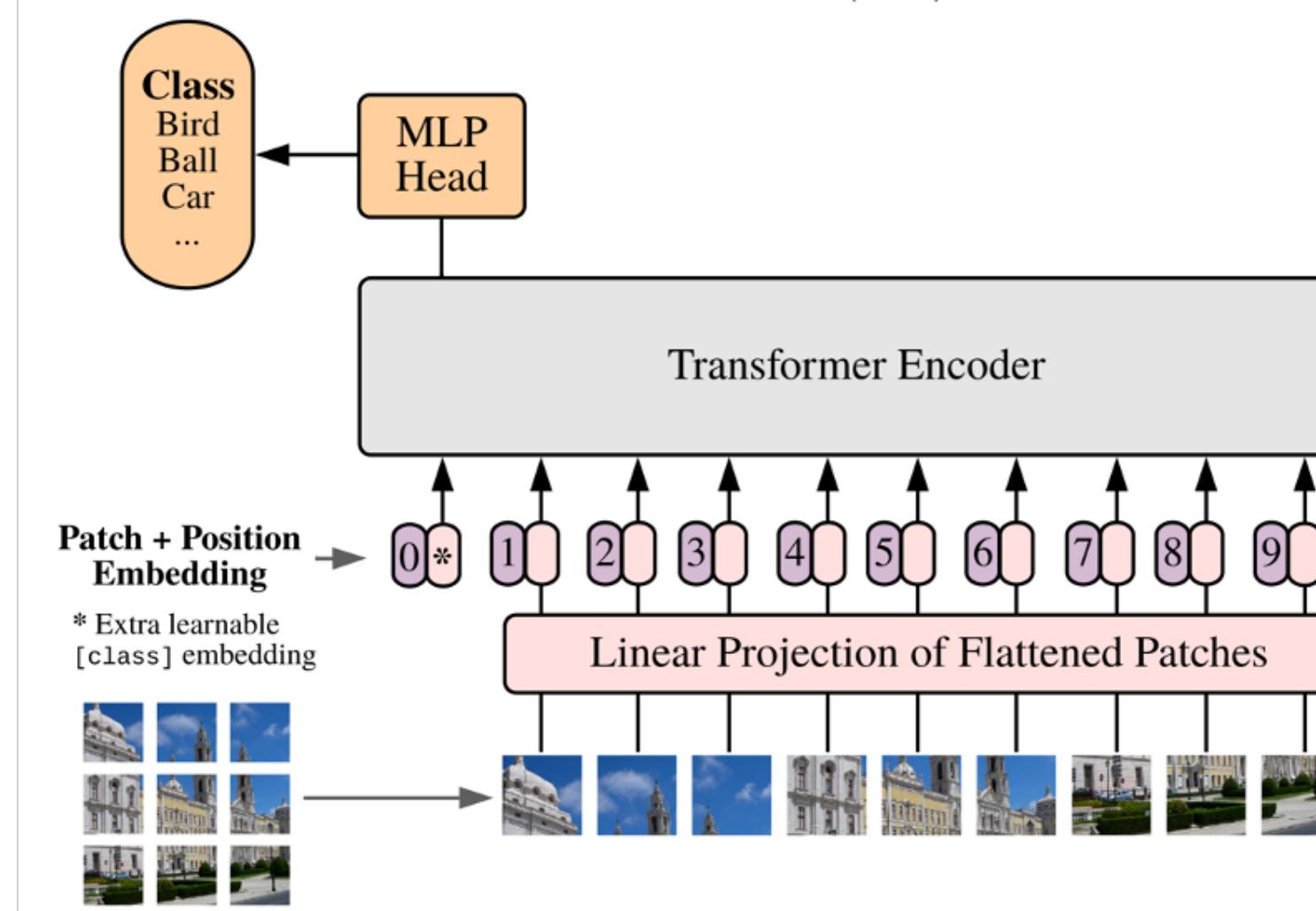
 **Leak, Cheat, Repeat: Data Contamination and Evaluation Malpractices in Closed-Source LLMs**

**Exciting  
Opportunities Ahead**

# Transformers

## Computer Vision

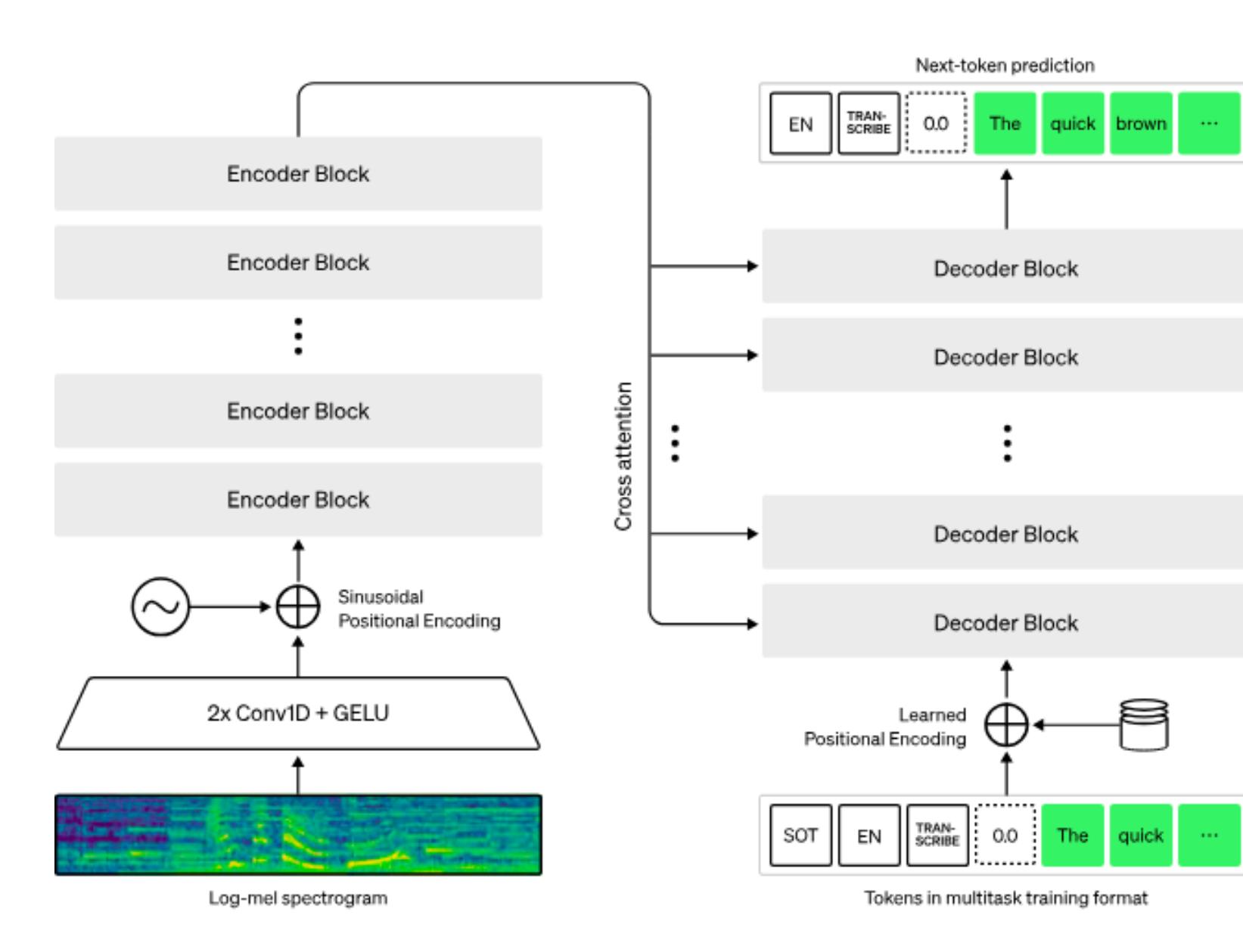
- ViT: cut image to patches
- Project each patch to a vector
- Treat them as token embeddings



# Transformers

## Speech

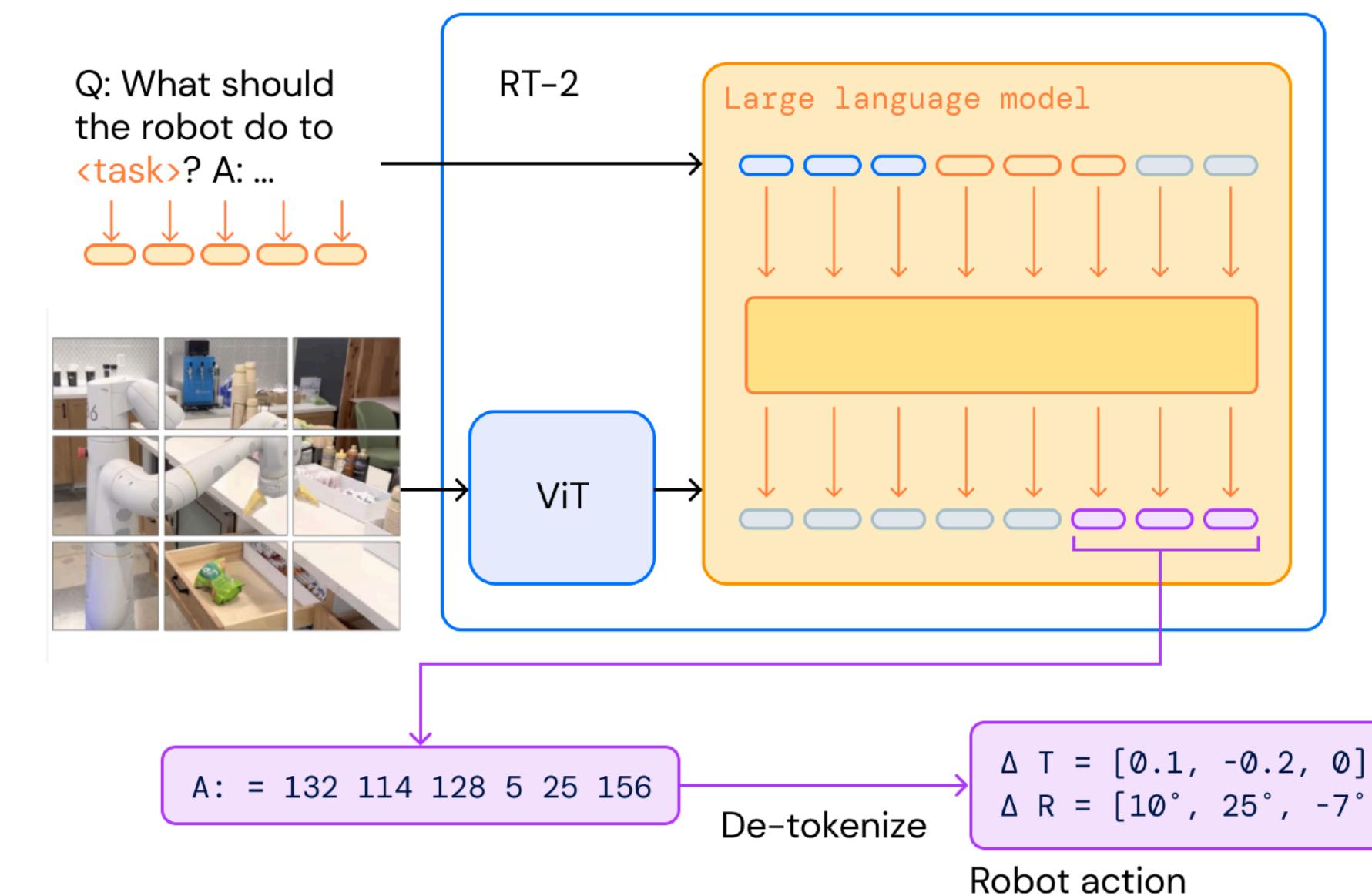
- Same as computer vision
- But: spectrograms instead of images
- The Whisper model



# Transformers

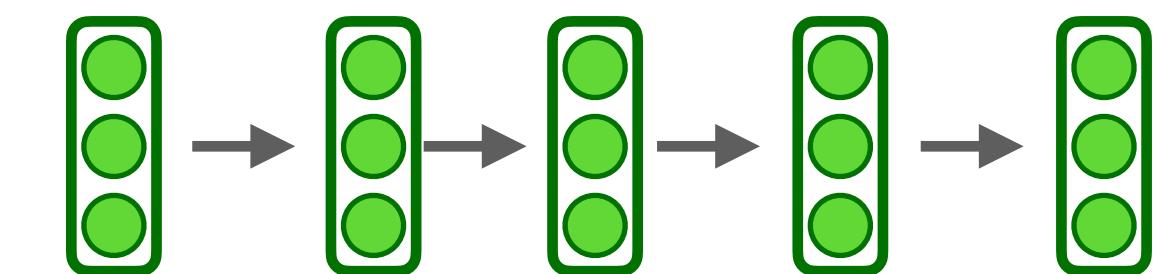
## Robotics

- Take observations and commands, all tokenized
- Output continuous joint control actions



# Today's Lecture

- ▶ **Part I: Fundamentals**
  - ▶ Intro, Motivation & Short History
  - ▶ Language Models (n-grams, FFNN-LM, Recap: FFNN)
- ▶ **Part II: Representations & Beyond FFNN**
  - ▶ RNNs (GRU/LSTMs), Attention
  - ▶ Contextualised Representations (ELMo)
- ▶ **Part III: Transformer & LLMs**
  - ▶ The Transformer, Masked LMs (BERT), Pre-training & Fine-tuning
  - ▶ Prompting, LLMs & Caution



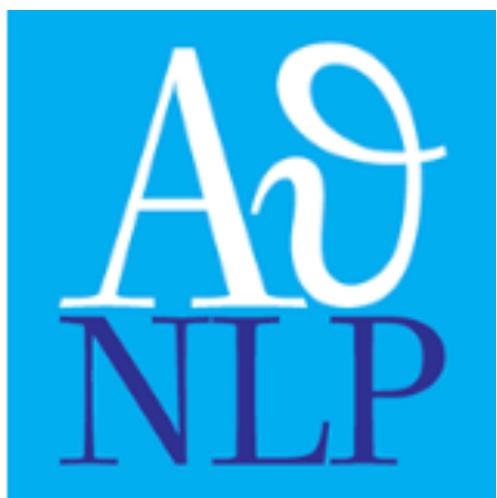


LUDWIG-  
MAXIMILIANS-  
UNIVERSITÄT  
MÜNCHEN

IT UNIVERSITY OF COPENHAGEN

# Questions? Thank you

Thanks to all the  
organizers & sponsors of:



Barbara Plank  
@barbara\_plank  
LMU Munich

[bplank.github.io](https://bplank.github.io)

# Core References

- Jurasky & Martin [SLP textbook](#) chapter 3 (n-gram LMs), 7 (neural LMs) & 8 (RNNs)
- Arianna Bisazza's [AthNLP 2019 lecture](#) on MT/Transformers
- Graham Neubig's [Advanced NLP class](#)
- Yoav Artzi's [LM-class](#) (also based on Greg Durrett's class material)
- Yoav Goldberg (2015): [A Primer on Neural Network Models for Natural Language Processing](#)
- Chris Manning & Abigail See (2018) Stanford NLP class
- Lilian Weng's [attention tutorial](#)