

# Introduction to Artificial Intelligence

IFT3335 Lecture: Natural Language Processing -  
part 2

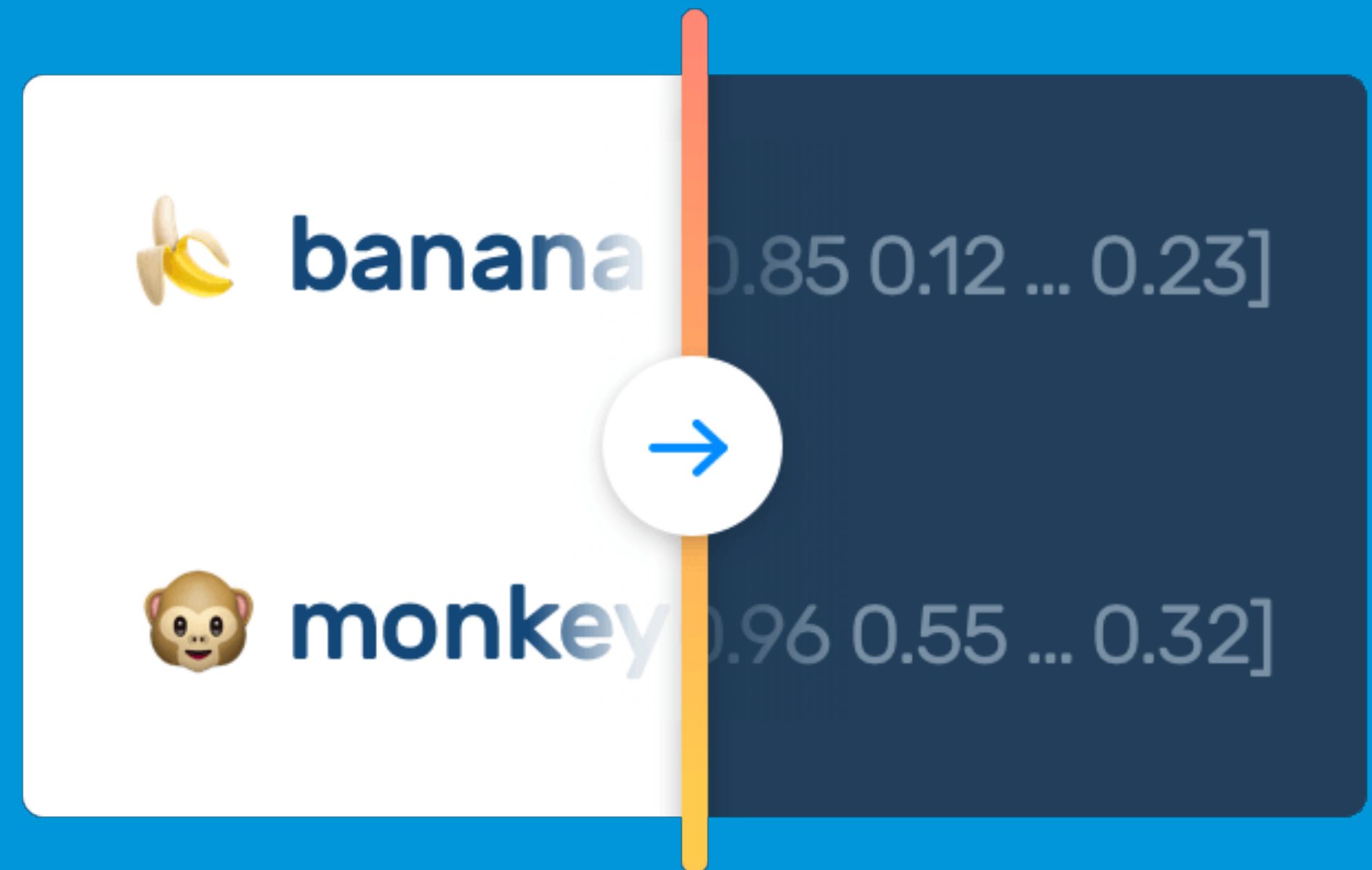
*Bang Liu, Jian-Yun Nie*



## 2 Lecture outline

1. Represent the Meaning of a Word: Word Embedding
2. Machine Translation: Sequence to Sequence and Attention
3. Transformer
4. BERT
5. AI: Research Frontiers

# Represent the Meaning of a Word: Word Embedding



## 4 Meaning Representations

- ◎ Definition of “Meaning”
  - the idea that is represented by a word, phrase, etc.
  - the idea that a person wants to express by using words, signs, etc.
  - the idea that is expressed in a work of writing, art, etc.



## 5 Corpus-Based Representations

- Atomic symbols: **one-hot** representation

car [0 0 0 0 0 0 1 0 0 ... 0]

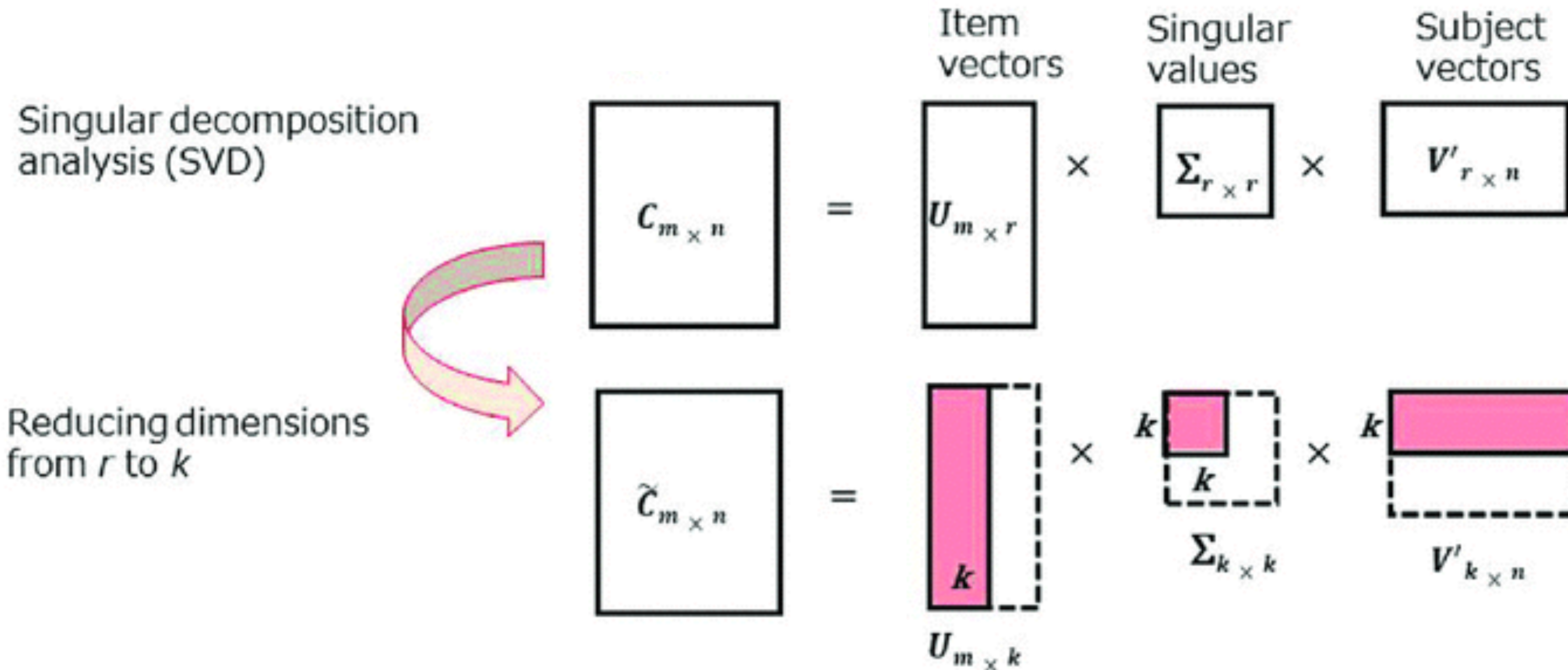
motorcycle [0 0 1 0 0 0 0 0 0 ... 0]

Issues: difficult to compute the similarity (i.e. comparing “car” and “motorcycle”)

Idea: words with similar meanings often have similar neighbors

## 6 Low-Dimensional Dense Word Vector

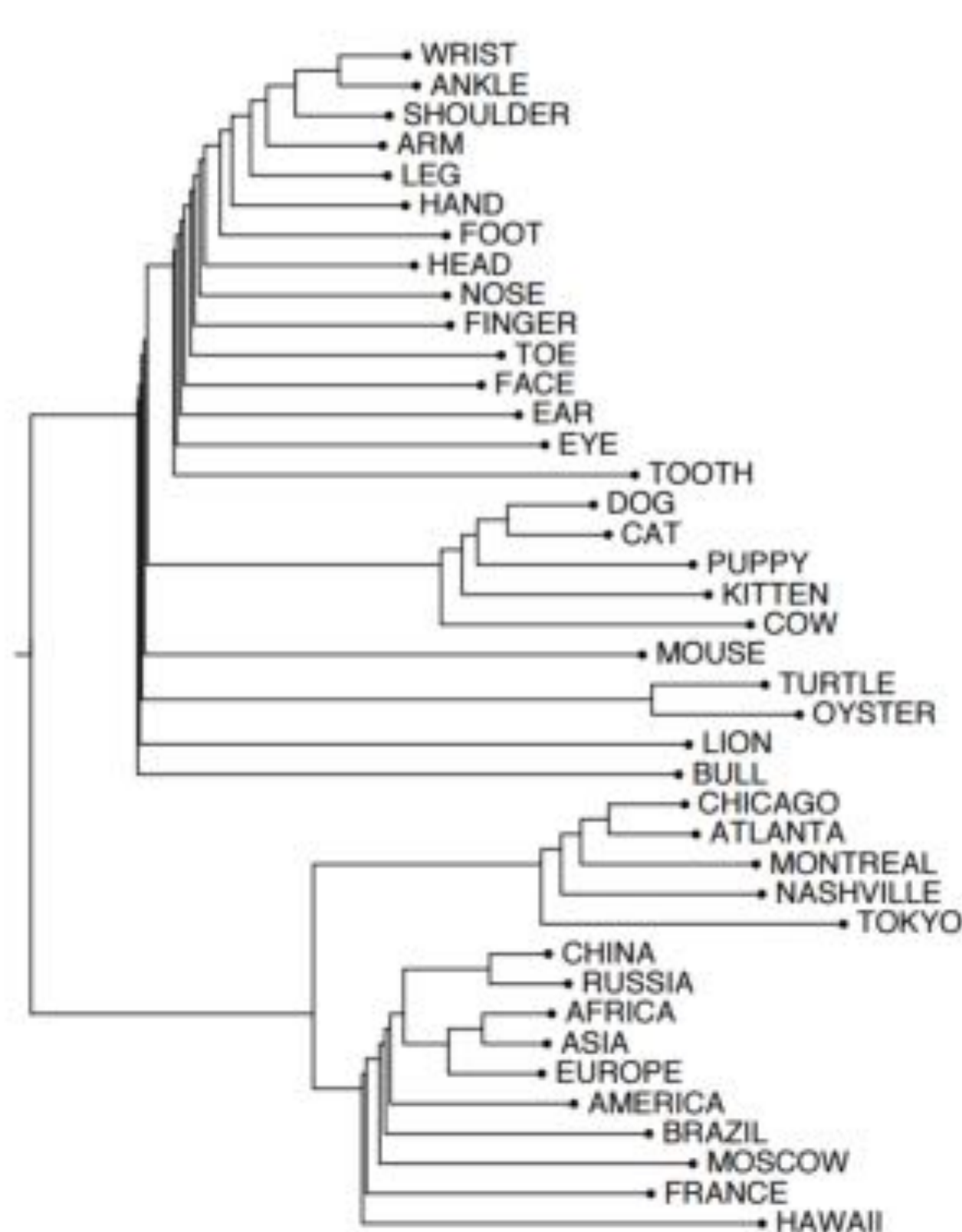
- Method 1: dimension reduction on the matrix
- Singular Value Decomposition (SVD) of co-occurrence matrix  $X$



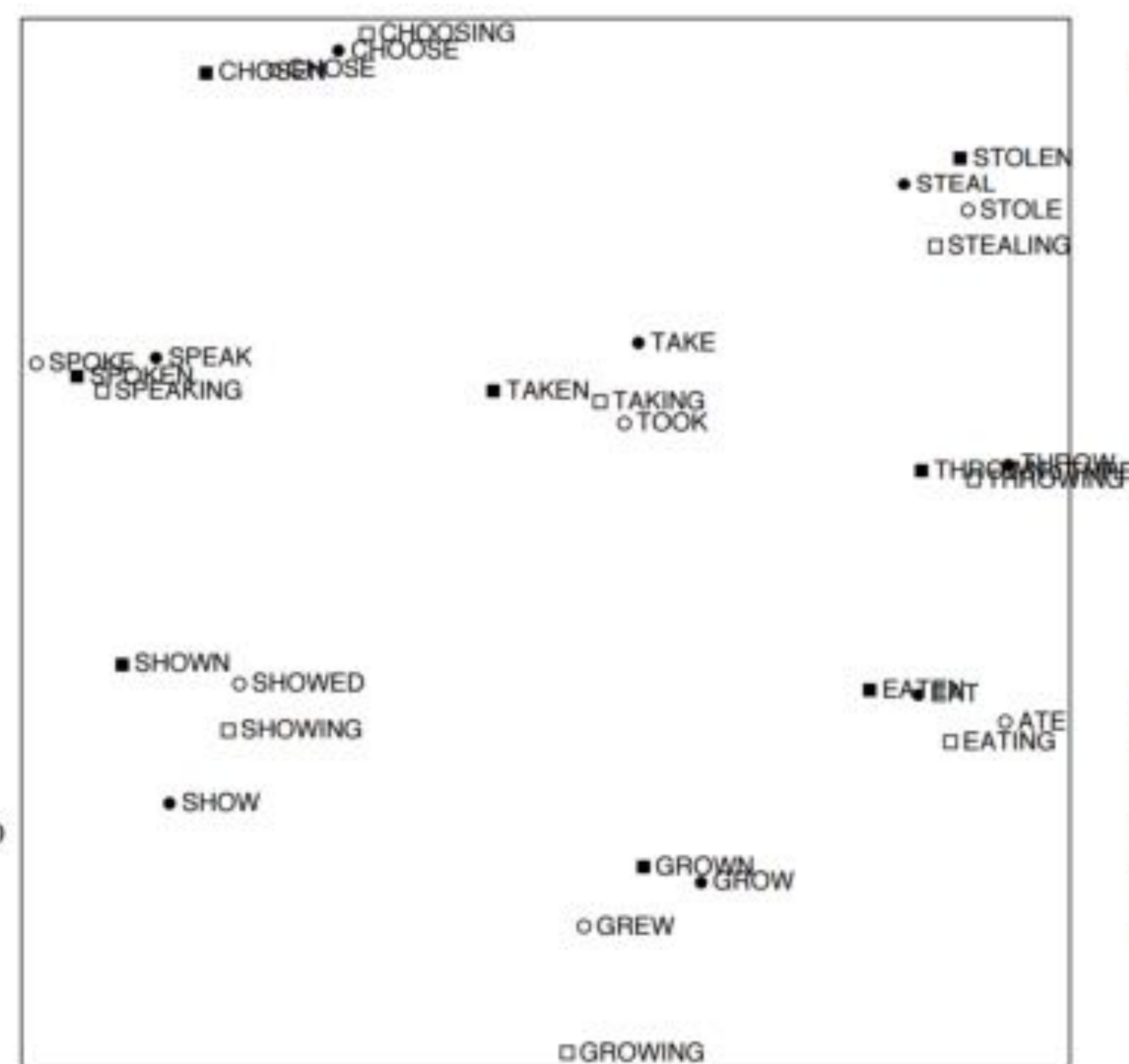


# 7 Low-Dimensional Dense Word Vector

- Method 1: dimension reduction on the matrix
- Singular Value Decomposition (SVD) of co-occurrence matrix  $X$



semantic relations



syntactic relations

Issues:

- computationally expensive:  
 $O(mn^2)$  when  $n < m$  for  $n \times m$  matrix
- difficult to add new words

Idea: directly learn low-dimensional word vectors

## 8 Low-Dimensional Dense Word Vector

- ◎ Method 2: directly learn low-dimensional word vectors
  - Learning representations by back-propagation. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - Most popular models: word2vec (Mikolov et al. 2013) and Glove (Pennington et al., 2014) (as known as “Word Embeddings ”)



**What are you like?**  
**Personality Embedding**

## 10 Big Five Personality Trait Test

- On a scale of 0 to 100, how introverted/extraverted are you (where 0 is the most introverted, and 100 is the most extraverted)?

Openness to experience	79	out	of	100
Agreeableness	75	out	of	100
Conscientiousness	42	out	of	100
Negative emotionality	50	out	of	100
Extraversion	58	out	of	100

Example of the result of a Big Five Personality Trait test. It can really tell you a lot about yourself and is shown to have predictive ability in [academic](#), [personal](#), and [professional success](#).



# 11 Central Idea: Represent Things by Vectors

1- We can represent things (and people) as vectors of numbers (Which is great for machines!)

Jay	-0.4	0.8	0.5	-0.2	0.3
-----	------	-----	-----	------	-----

2- We can easily calculate how similar vectors are to each other

The people most similar to Jay are:

cosine\_similarity ▼

Person #1	0.86
Person #2	0.5
Person #3	-0.20

**Words can also be  
Represented by Vectors:  
Word Embedding**

## 13 Word Embedding

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 ,  
-0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,  
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 ,  
0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 ,  
0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 ,  
-2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 ,  
1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 ,  
0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 ,  
0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 ,  
0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

“King”

This is a word embedding for the word “king” (GloVe vector trained on Wikipedia).

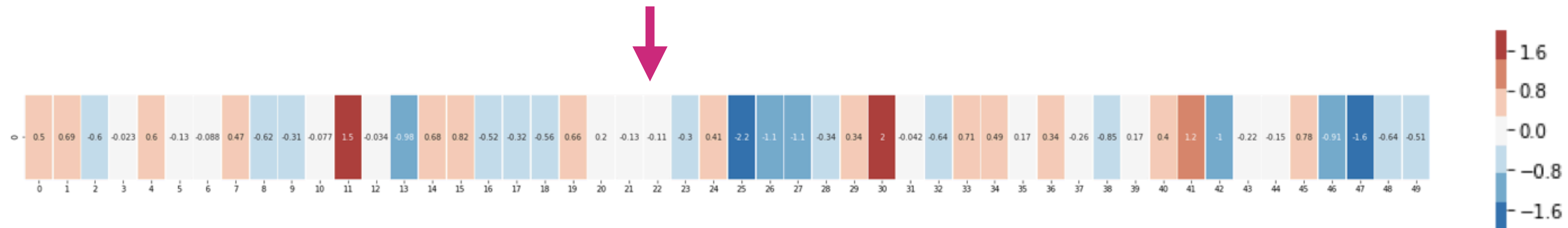


# Visualize Word Embedding

- Let's color code the cells based on their values (red if they're close to 2, white if they're close to 0, blue if they're close to -2)

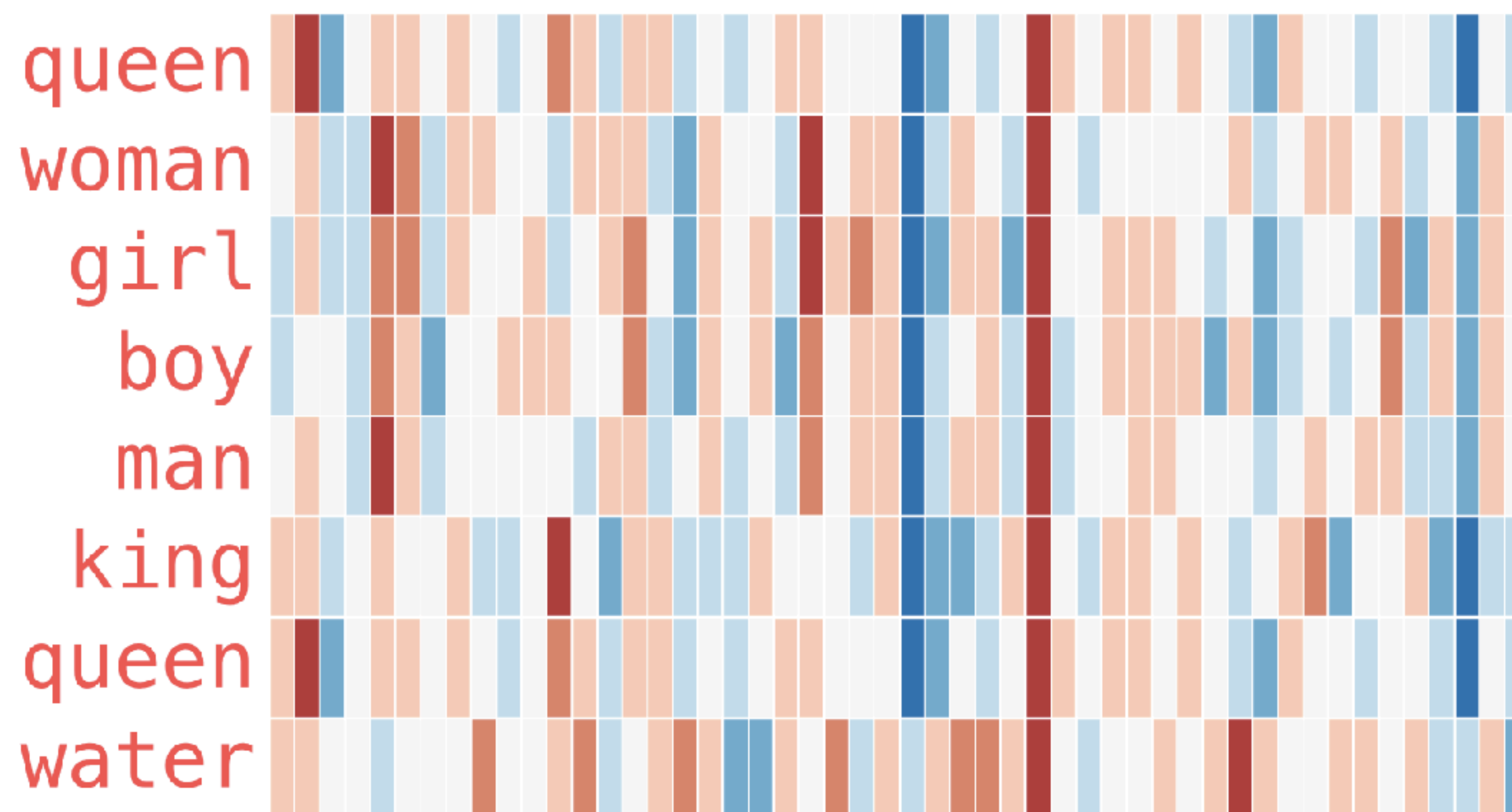
```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 ,  
-0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,  
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 ,  
0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 ,  
0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 ,  
-2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 ,  
1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 ,  
0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 ,  
0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 ,  
0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

“King”

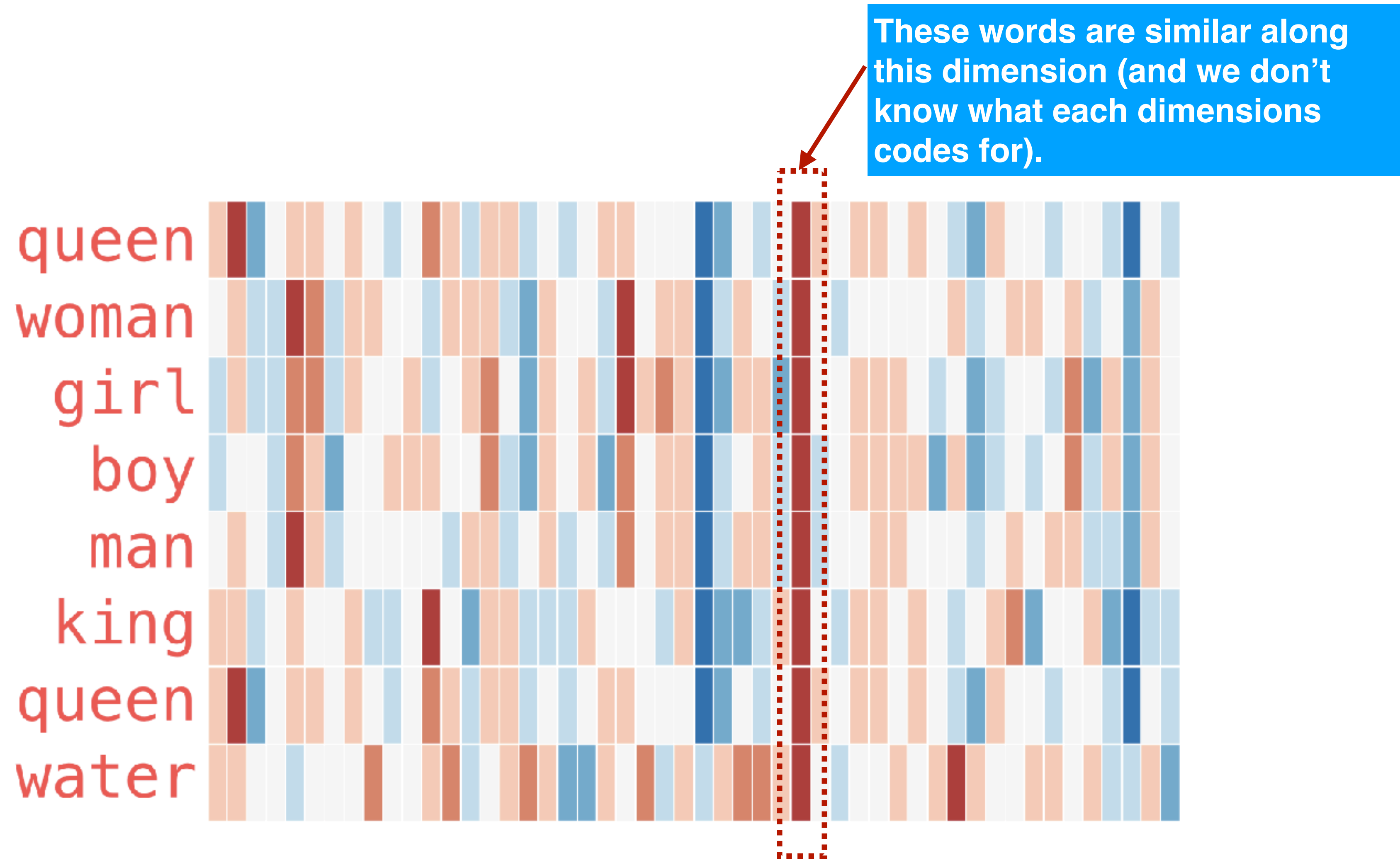


# 15 Compare Word Embeddings

- A list of examples (compare by vertically scanning the columns looking for columns with similar colors).



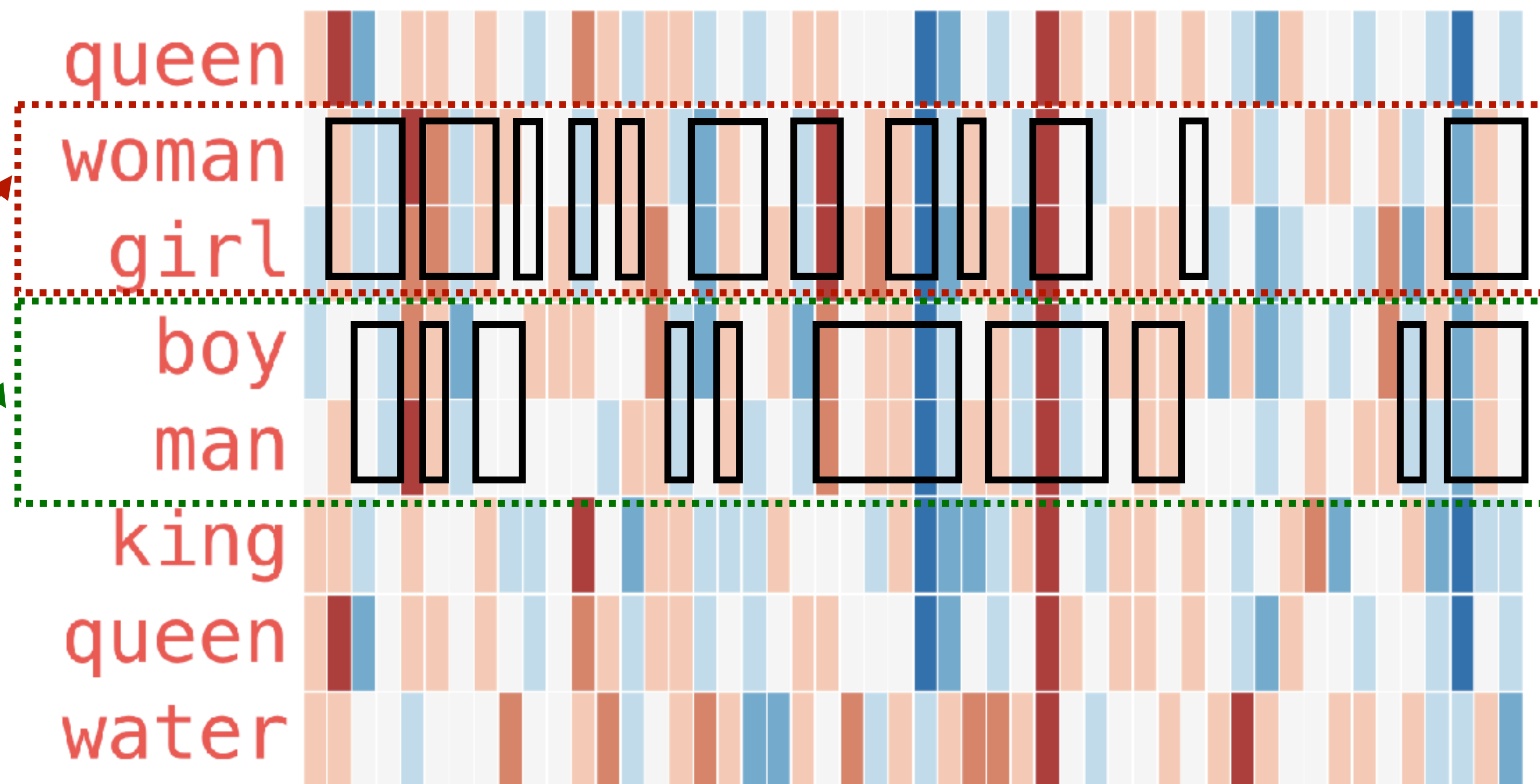
# Compare Word Embeddings





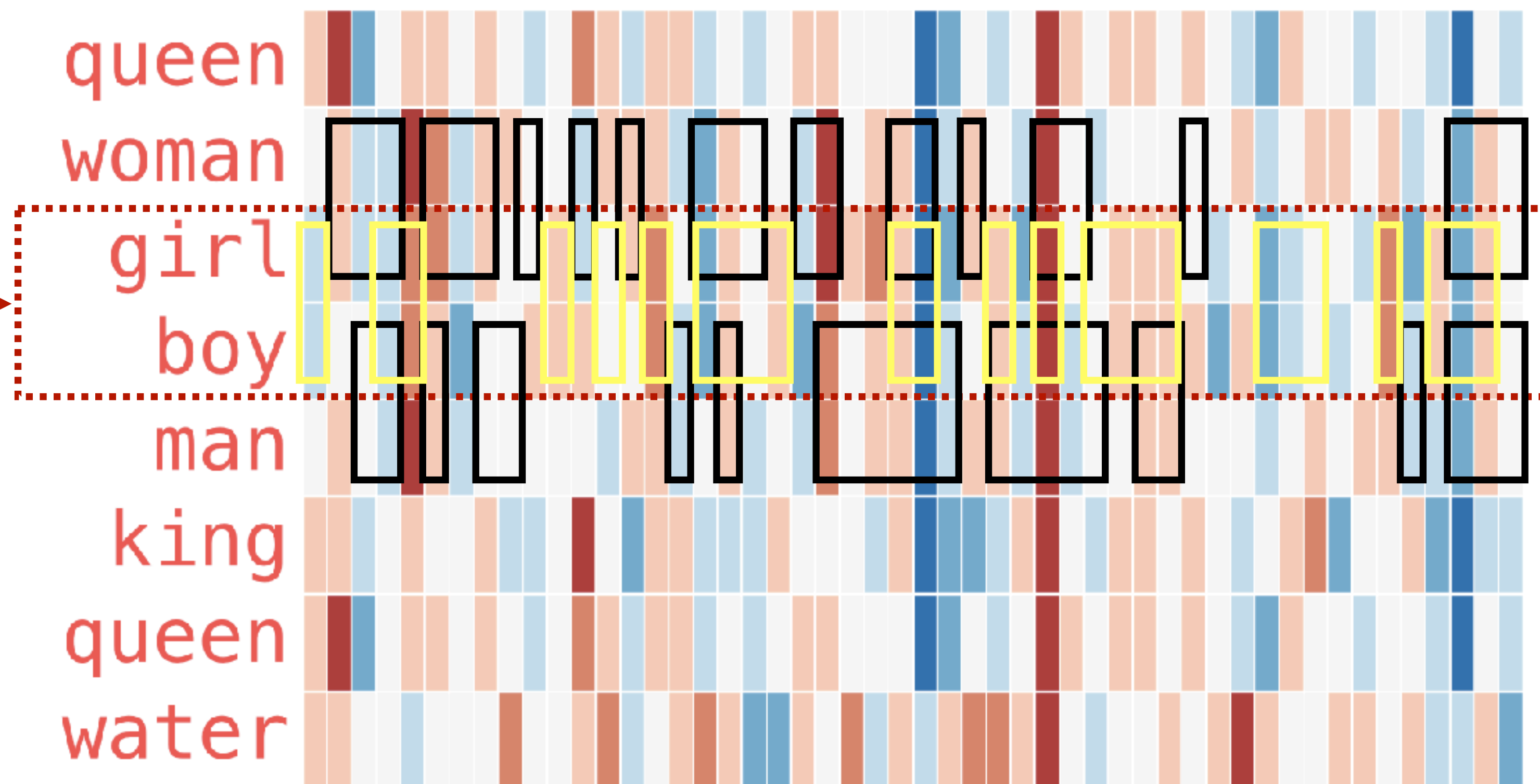
# 17 Compare Word Embeddings

“woman” and “girl” are similar to each other in a lot of places. The same with “man” and “boy”.



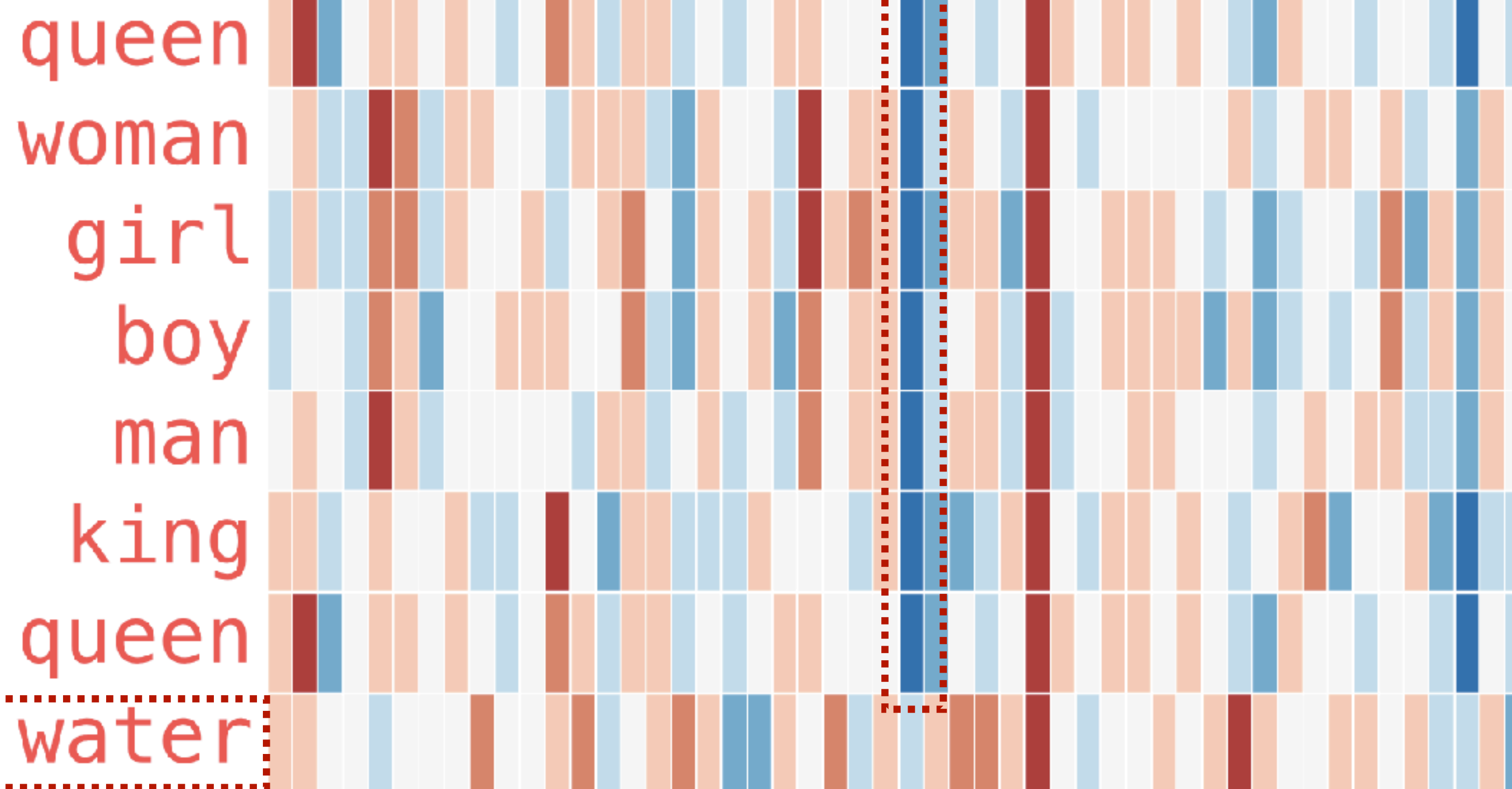
# 18 Compare Word Embeddings

“boy” and “girl” also have places where they are similar to each other, but different from “woman” or “man”. Could these be coding for a vague conception of youth? possible.



# 19 Compare Word Embeddings

Different category!

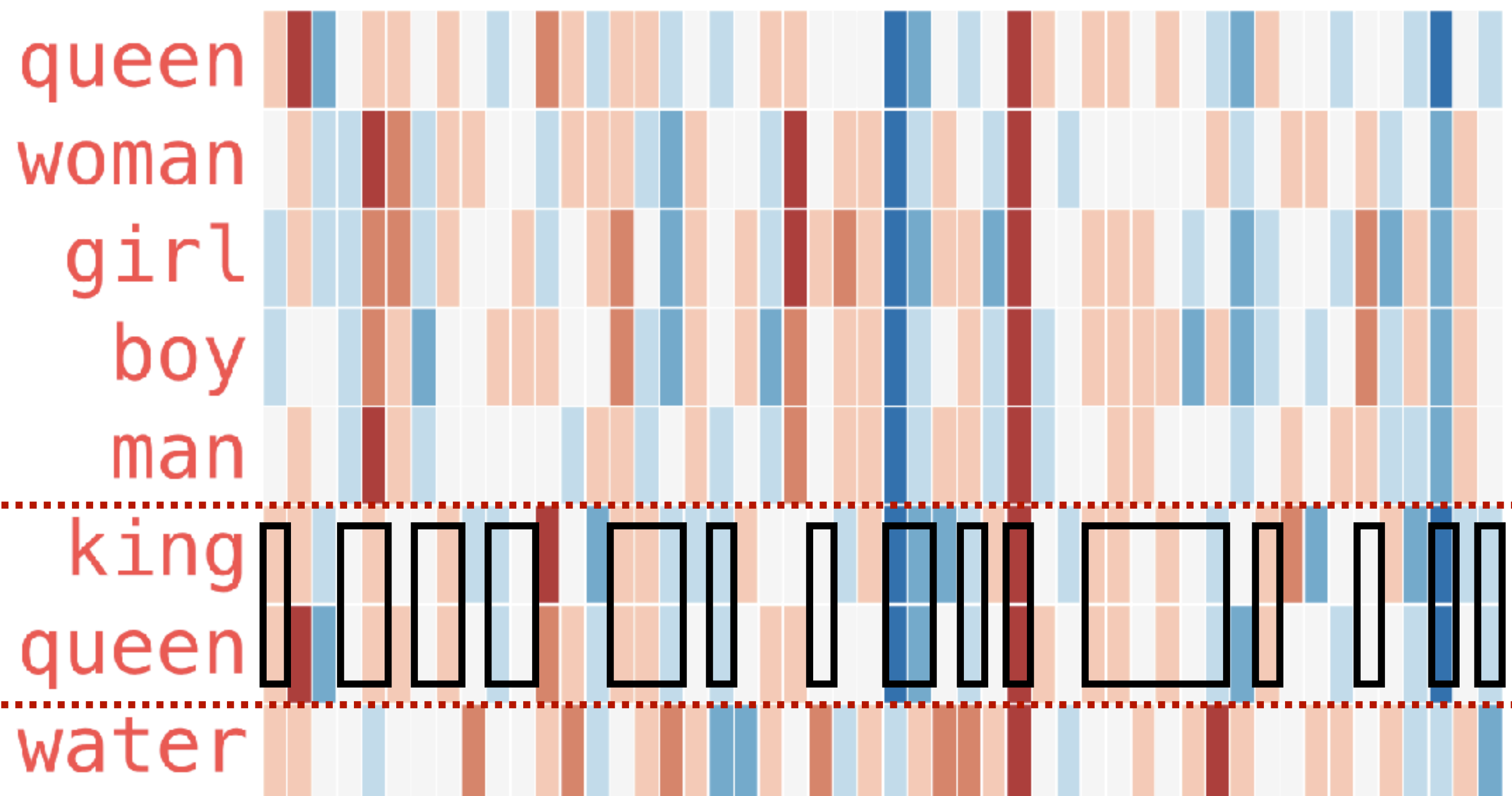


This column goes all the way down and stops before the embedding for "water".



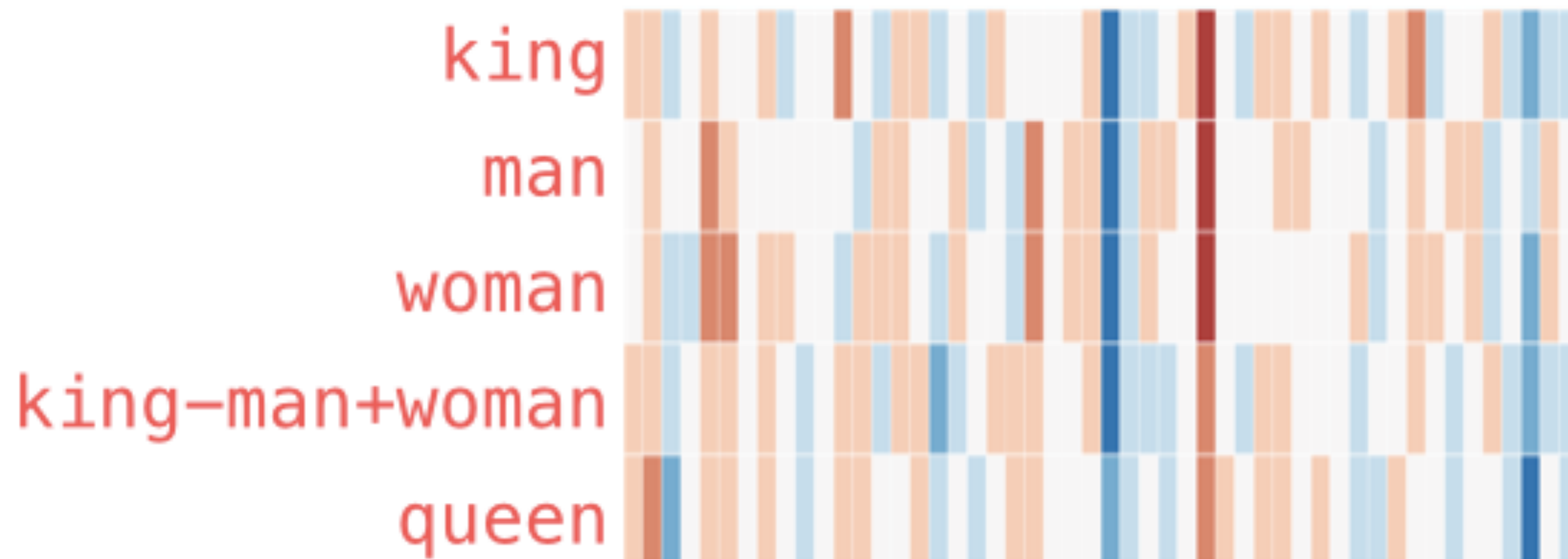
## 20 Compare Word Embeddings

There are clear places where “king” and “queen” are similar to each other and distinct from all the others. Could these be coding for a vague concept of royalty?



## 21 Compare Word Embeddings

king - man + woman ≈ queen



The resulting vector from "king-man+woman" doesn't exactly equal "queen", but "queen" is the closest word to it from the 400,000 word embeddings we have in this collection.

# Compare Word Embeddings

```
model.most_similar(positive=["king", "woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),  
 ('throne', 0.7664333581924438),  
 ('prince', 0.7592144012451172),  
 ('daughter', 0.7473883032798767),  
 ('elizabeth', 0.7460219860076904),  
 ('princess', 0.7424570322036743),  
 ('kingdom', 0.7337411642074585),  
 ('monarch', 0.721449077129364),  
 ('eldest', 0.7184862494468689),  
 ('widow', 0.7099430561065674)]
```

Using the Gensim library in python, we can add and subtract word vectors, and it would find the most similar words to the resulting vector. The image shows a list of the most similar words with “king+woman-man”, each with its cosine similarity.



# **How to Train Word Embeddings?**

## **Recall Language Modeling**

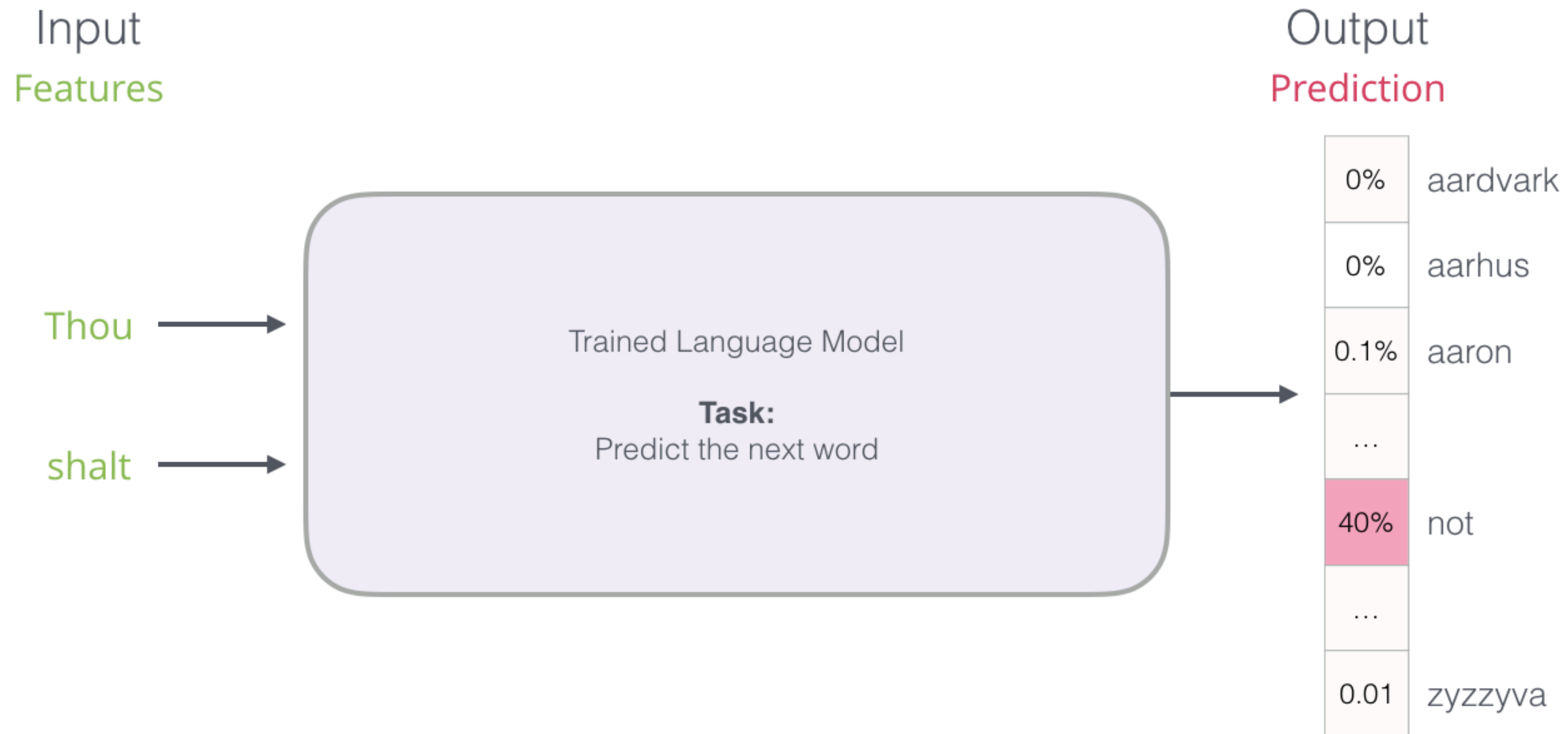
## 24 Recall Language Modeling

- A **language model** can take a list of words (let's say two words), and attempt to predict the word that follows them.



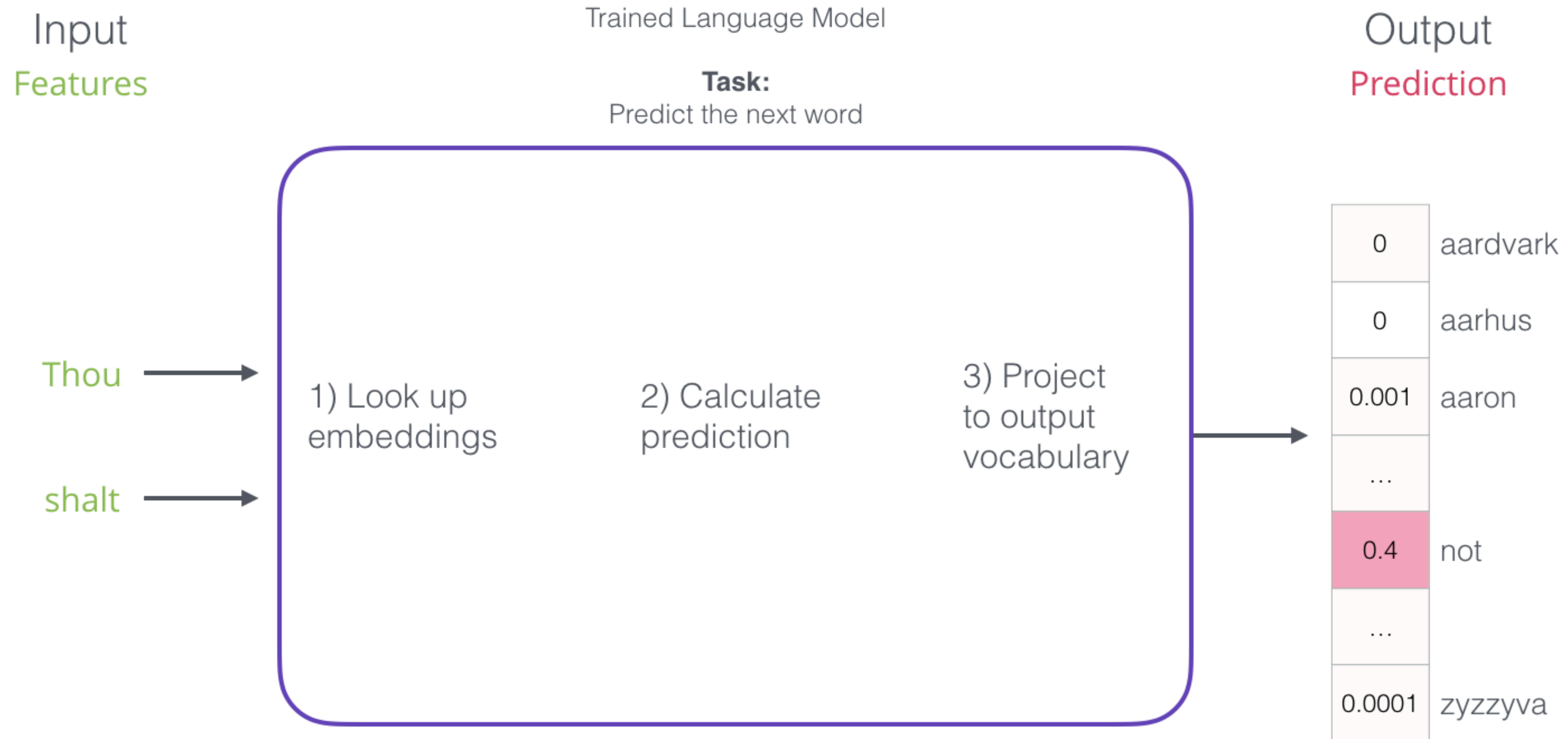
## 25 Recall Language Modeling

- A language model actually outputs a probability score for all the words it knows (the model's "vocabulary")



# Recall Language Modeling

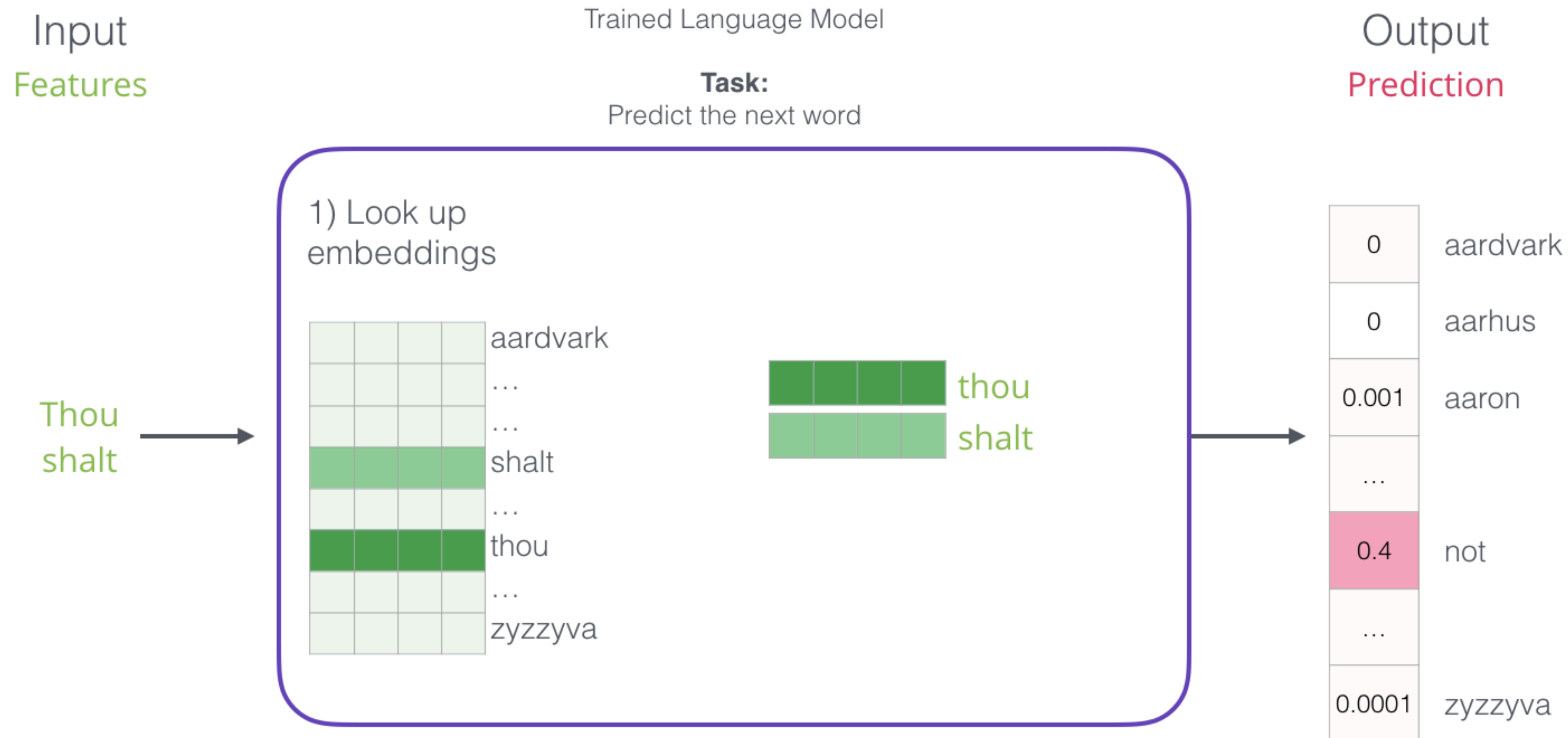
- After being trained, early neural language models (Bengio 2003) would calculate a prediction in three steps:





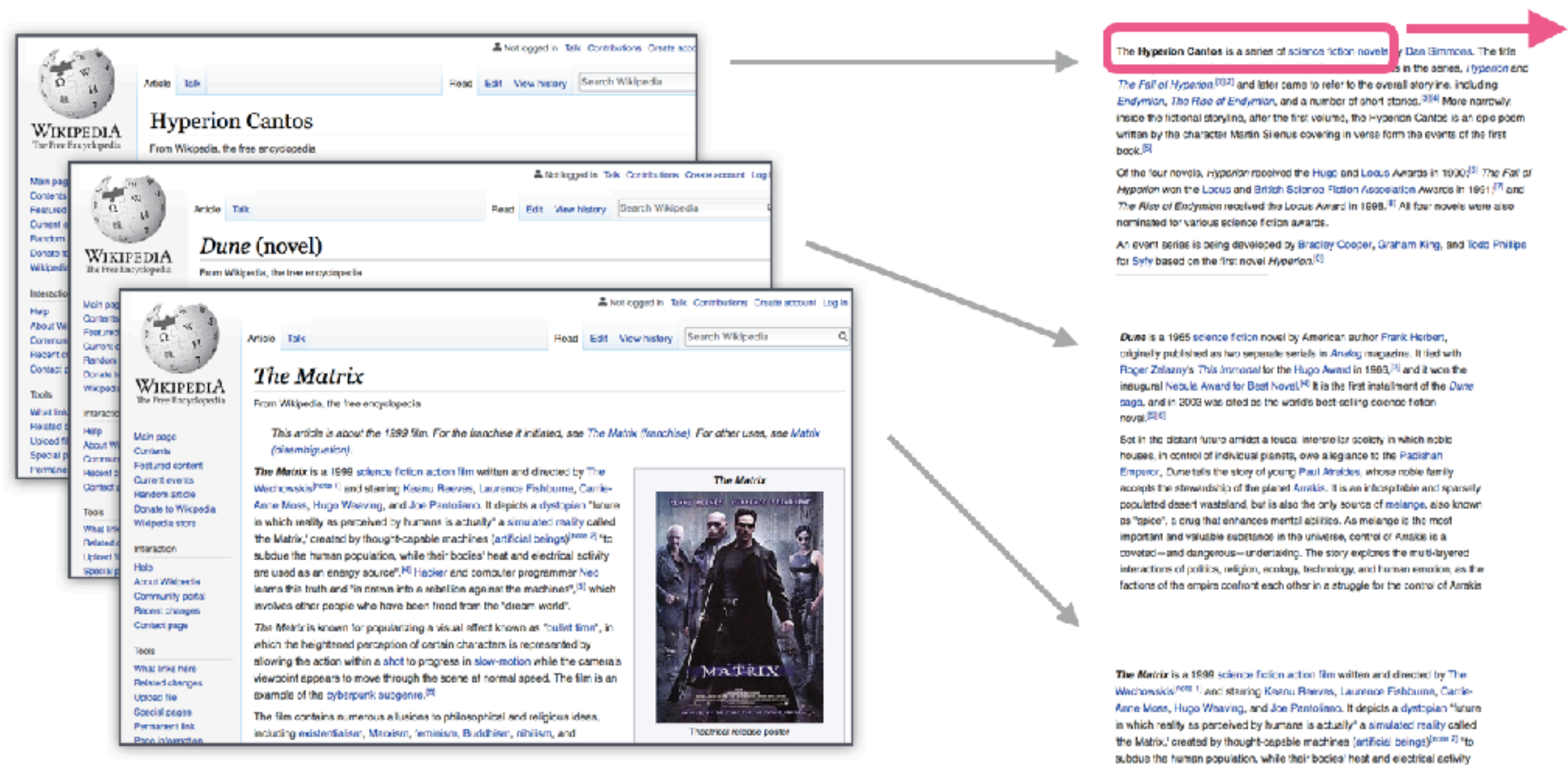
# 27 Recall Language Modeling

- In the first step, we get a **matrix that contains an embedding for each word** in our vocabulary.



# Language Model Training

- Words get their embeddings by looking at which other words they tend to appear next to.
  - We get a lot of text data (say, all Wikipedia articles, for example).
  - We have a window (say, of three words) that we slide against all of that text.
  - The sliding window generates training samples for our model



**We take the first two words to be features, and the third word to be a label:**



Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

Dataset

input 1	input 2	output
thou	shalt	not

**We then slide our window to the next position and create a second sample:**



Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	...
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make

**And pretty soon we have a larger dataset of which words tend to appear after different pairs of words:**



Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	...
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

**From Language Modeling to  
Word Embedding:  
Look Both Ways**



## 31 Language Model Training

Jay was hit by a \_\_\_\_\_



## Look Both Ways

Jay was hit by a \_\_\_\_\_



“bus”

Jay was hit by a \_\_\_\_\_ bus



?

# Word2Vec: CBOW and Skip-gram

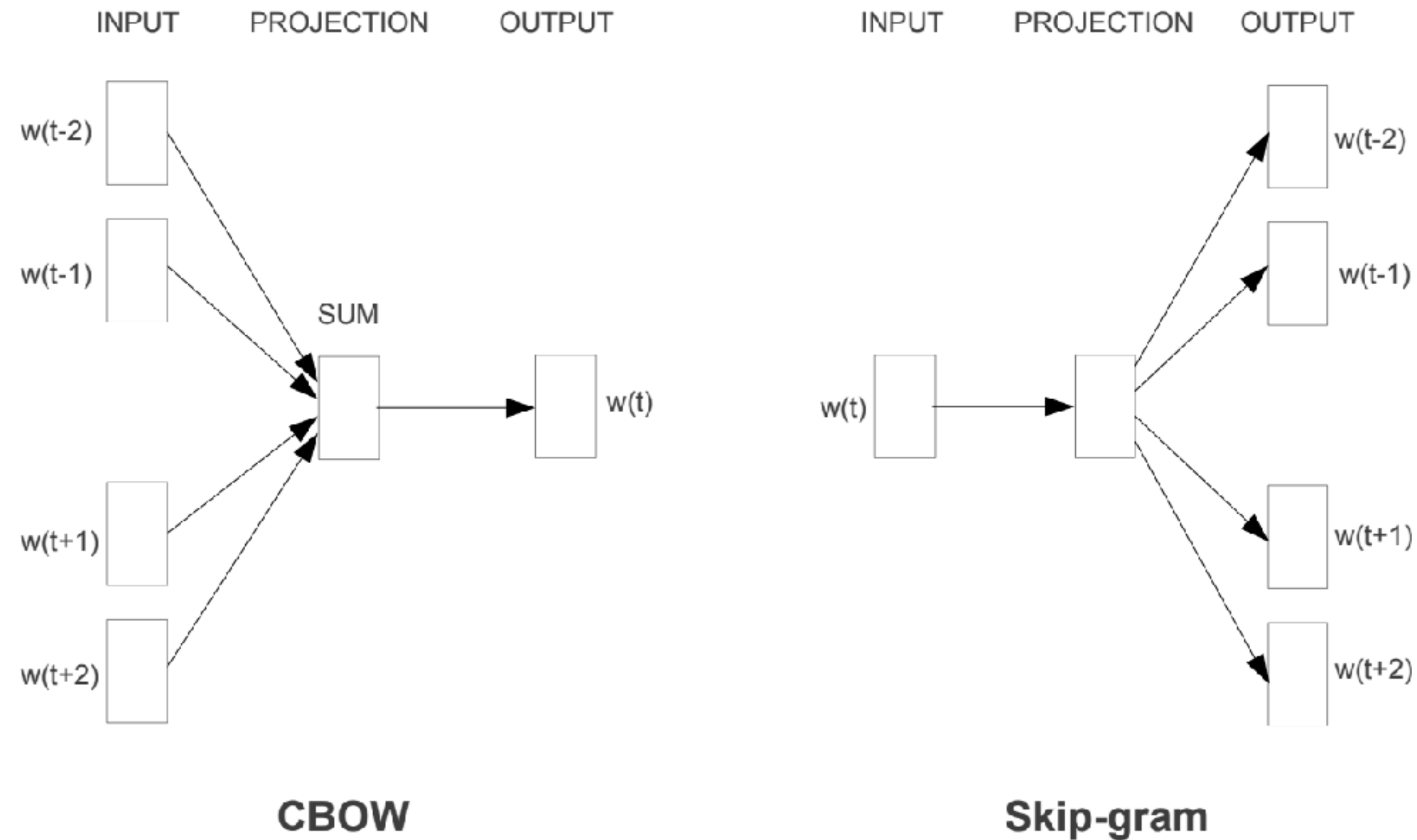


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

# CBOW: Continuous Bag of Words

- Instead of only looking at words before the target word, we can also look at words after it.

*“You shall know a word by the company it keeps” — J.R. Firth*

Jay was hit by a \_\_\_\_\_ bus in...

by	a	red	bus	in
----	---	-----	-----	----



**Build training dataset**

input 1	input 2	input 3	input 4	output
by	a	bus	in	red



## 35 Skip-gram

- Instead of guessing a word based on its context (the words before and after it), this other architecture tries to guess neighboring words using the current word.

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a

The word in the green slot would be the input word, each pink box would be a possible output. The pink boxes are in different shades because this sliding window actually creates four separate samples in our training dataset.

# Skip-gram

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

By sliding our window to the next positions, a couple of positions later, we will have a lot more training examples.

Step 1: grab an example from the dataset. Feed it into an untrained model asking it to predict an appropriate neighbour word.

Step 2: The model conducts the three steps and outputs a prediction vector (with a probability assigned to each word in its vocabulary)

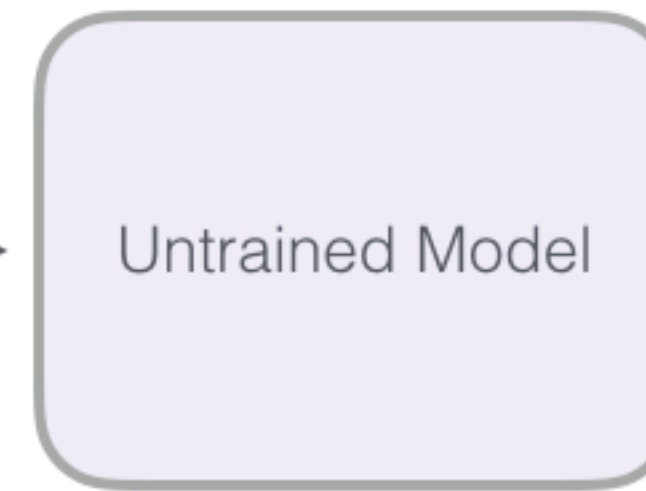
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Actual Target

0
0
0
...
0
1
...
0

- 1) Look up embeddings
- 2) Calculate prediction
- 3) Project to output vocabulary

not



Model Prediction

0	aardvark	=	0
0	aarhus		0
0.001	aaron		-0.001
...			...
0.4	taco		-0.4
0.001	thou		0.999
...			...
0.0001	zyzzyva		-0.0001

Error

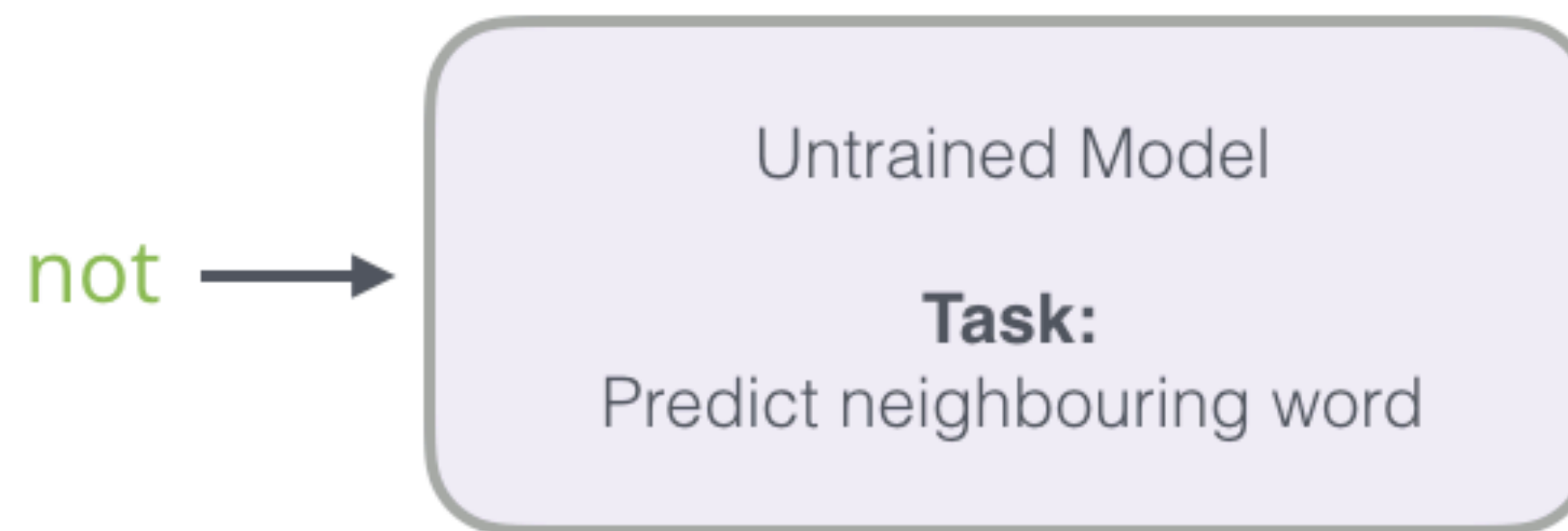
Update Model Parameters

Step 4: We proceed to do the same process with the next sample in our dataset, and then the next, until we've covered all the samples in the dataset.

Step 3: This error vector can now be used to update the model so the next time, it's a little more likely to guess thou when it gets not as input.

## 38 Negative Sampling

- The third step is very expensive from a computational point of view. **How to improve the performance?**



1) Look up embeddings

2) Calculate prediction

**3) Project to output vocabulary**

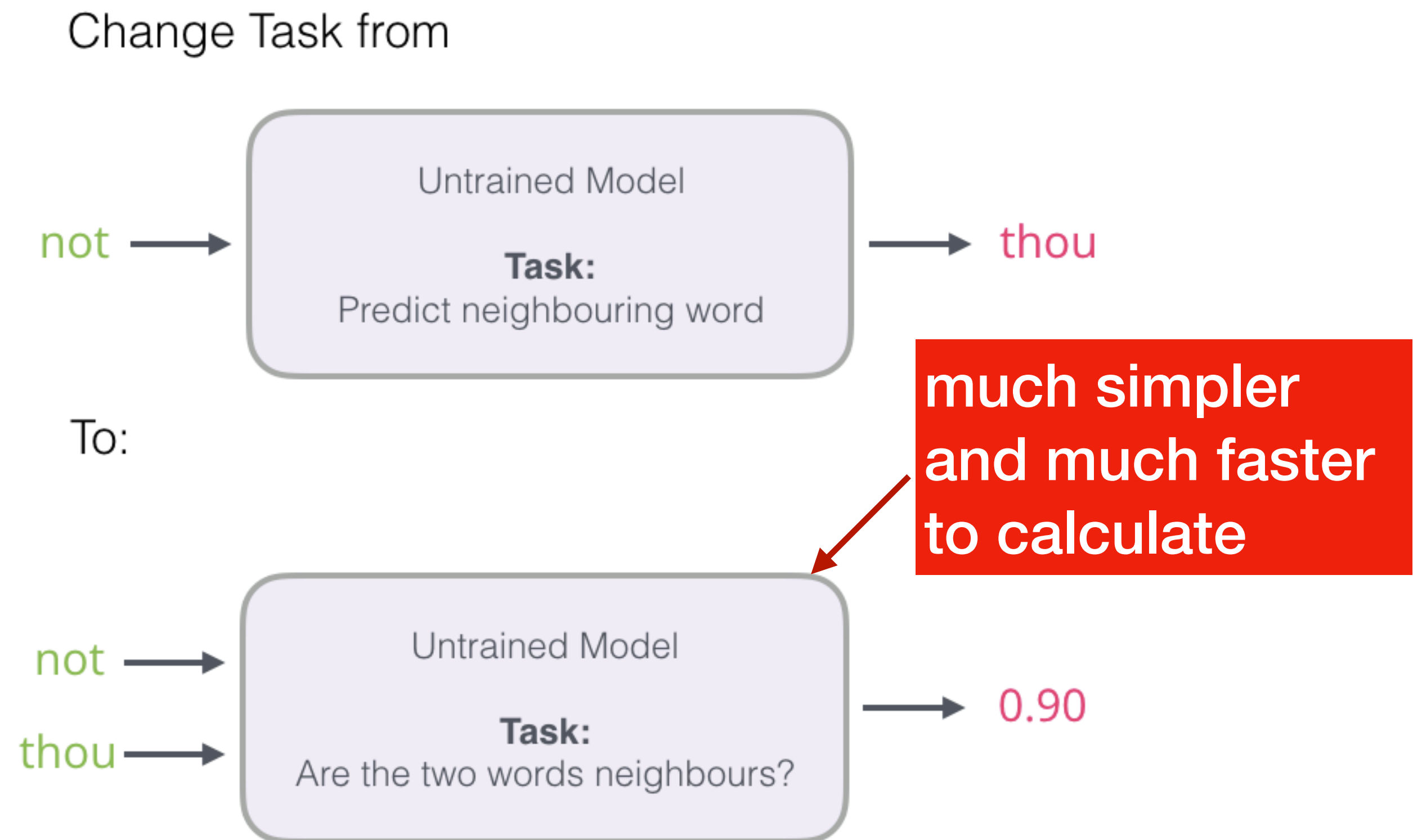
**[Computationally Intensive]**



# Negative Sampling

- One way is to split our target into two steps:
  1. Generate high-quality word embeddings (Don't worry about next-word prediction);
  2. Use these high-quality embeddings to train a language model (to do next-word prediction).

To **generate high-quality embeddings** using a high-performance model, we can switch the model's task from **predicting a neighboring word** to **takes the input and output word, and outputs a score indicating if they're neighbors or not** (0 for "not neighbors", 1 for "neighbors").



## 40 Negative Sampling

- Switch the structure of our dataset

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine



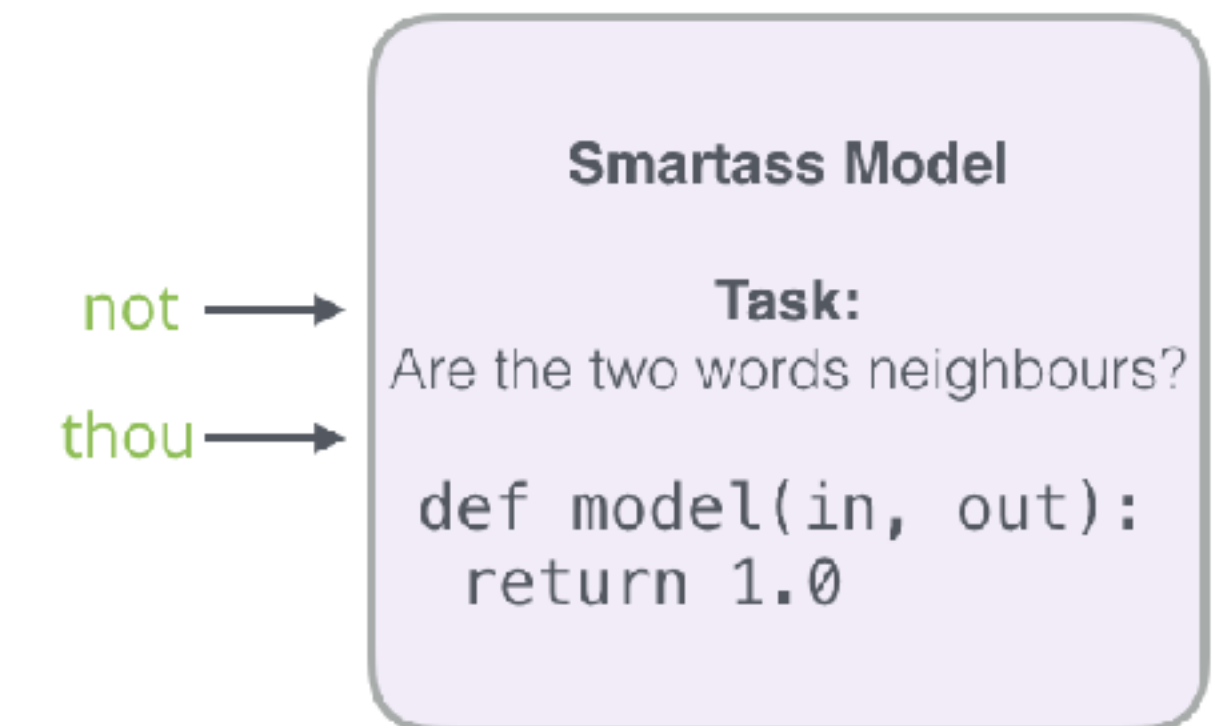
input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

# 41 Negative Sampling

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

No negative sample



So a smartass model that always returns 1 – achieving 100% accuracy, but learning nothing and generating garbage embeddings.

# 42 Skip-gram with Negative Sampling

- Randomly selected words that are not neighbors from the vocabulary to get **negative examples**.

Skipgram

shalt	not	make	a	machine
input		output		
make		shalt		
make		not		
make		a		
make		machine		

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Pick randomly from vocabulary (random sampling)

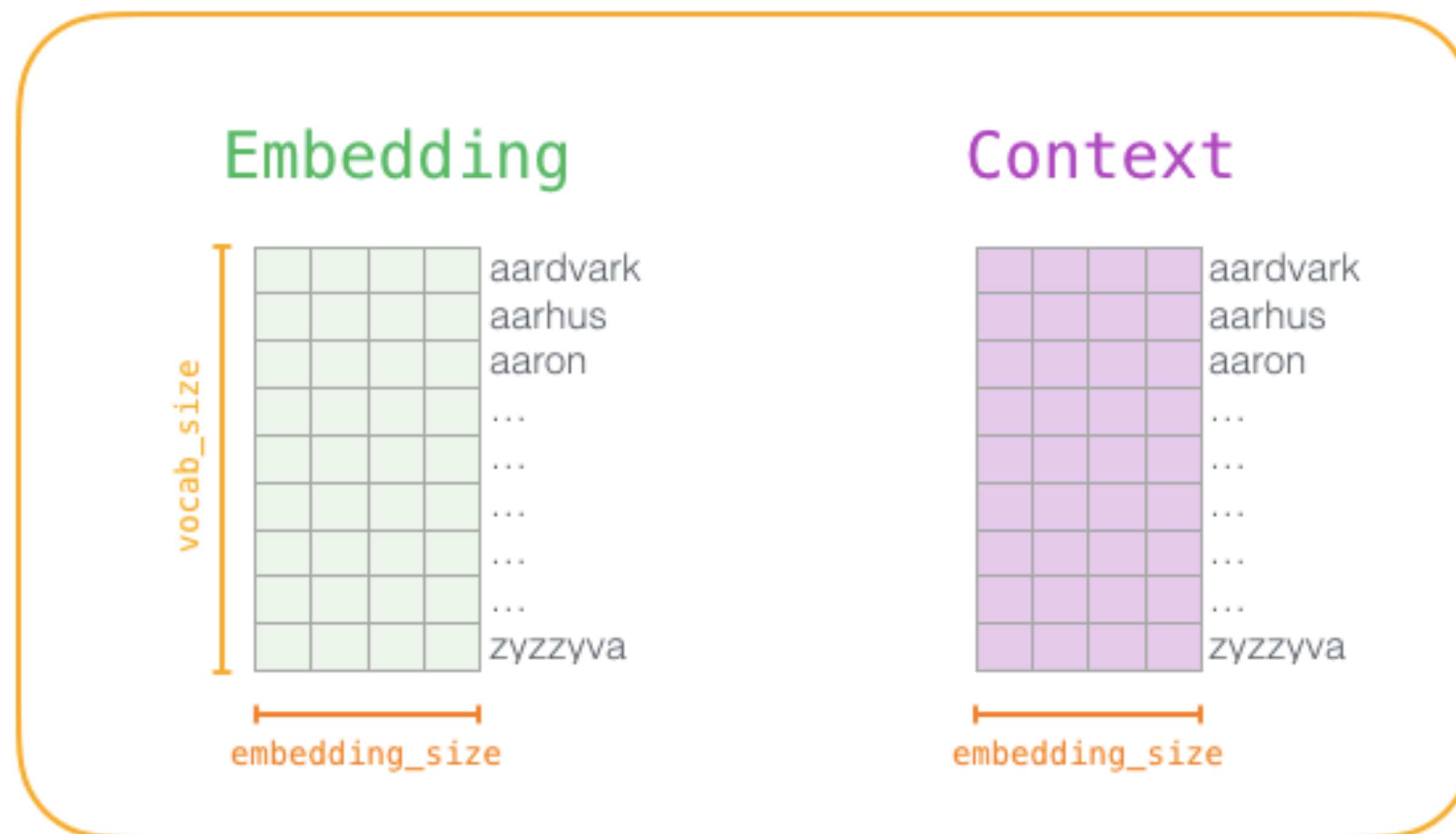
Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		



# Word2Vec Training Process

# Model initialization

- At the start of the training phase, we create two matrices – an **Embedding** matrix and a **Context** matrix. These two matrices have an embedding for each word in our vocabulary. We initialize these matrices with random values. (Why two vectors? Easier optimization.)

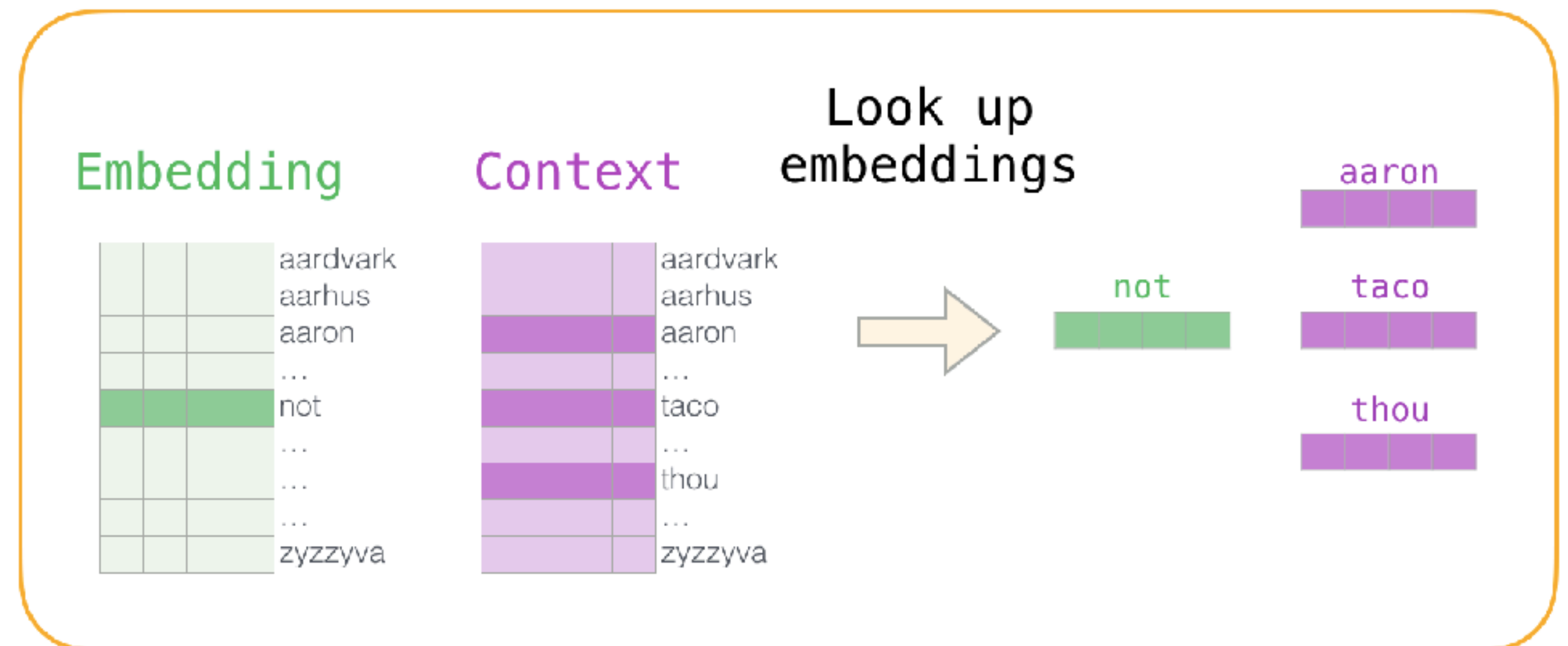


## 45 Feed Data

- For the input word, we look in the Embedding matrix. For the context words, we look in the Context matrix

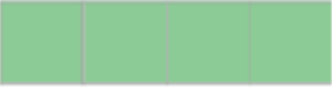





dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...



# Forward Propagation

- Take the dot product of the input embedding with each of the context embeddings.
- Calculate probability by Sigmoid().
- Calculate error.

input word	output word	target	input • output	sigmoid()	Error
not 	thou 	1	0.2	0.55	0.45
not 	aaron 	0	-1.11	0.25	-0.25
not 	taco 	0	0.74	0.68	-0.68

$$\text{error} = \text{target} - \text{sigmoid\_scores}$$

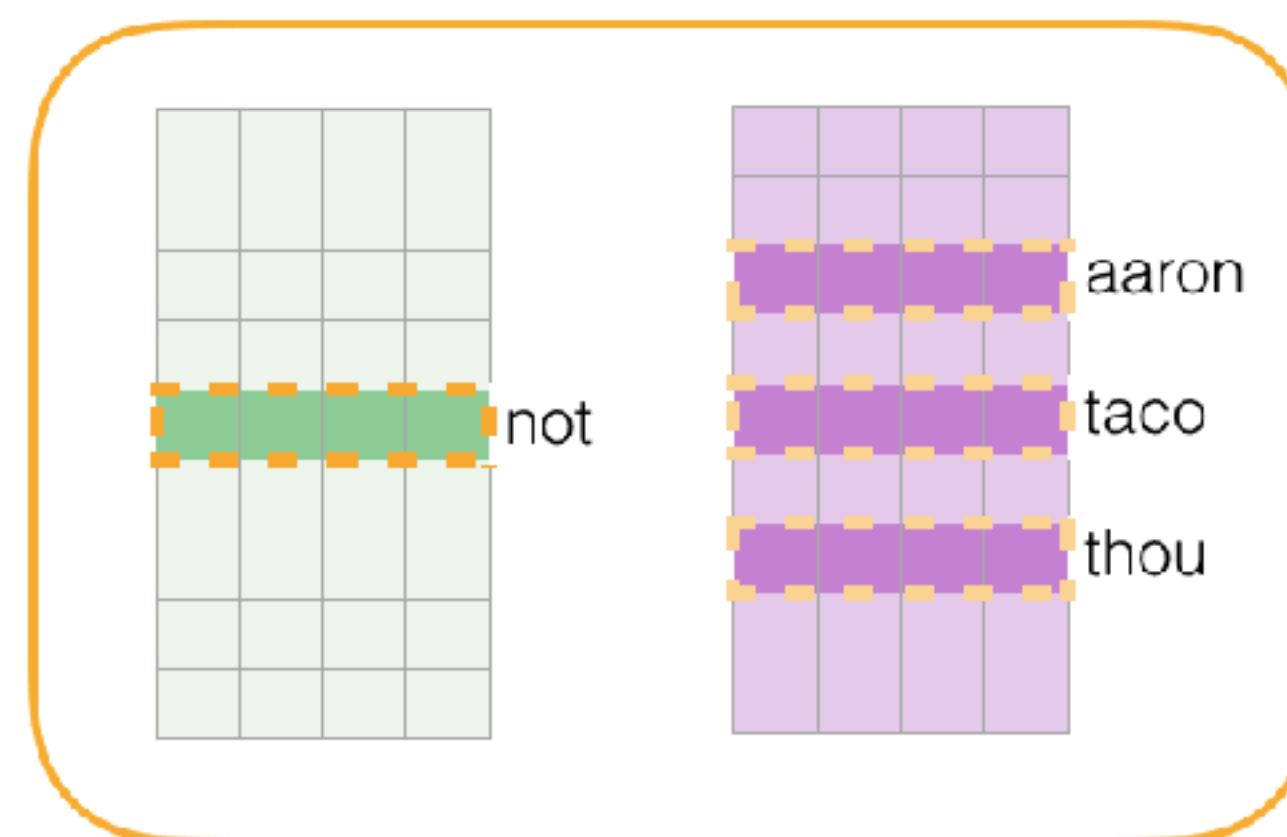


# 47 Back-propagation

- We can now use this error score to adjust the embeddings of **not**, **thou**, **aaron**, and **taco** so that the next time we make this calculation, the result would be closer to the target scores.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

$$\text{error} = \text{target} - \text{sigmoid\_scores}$$



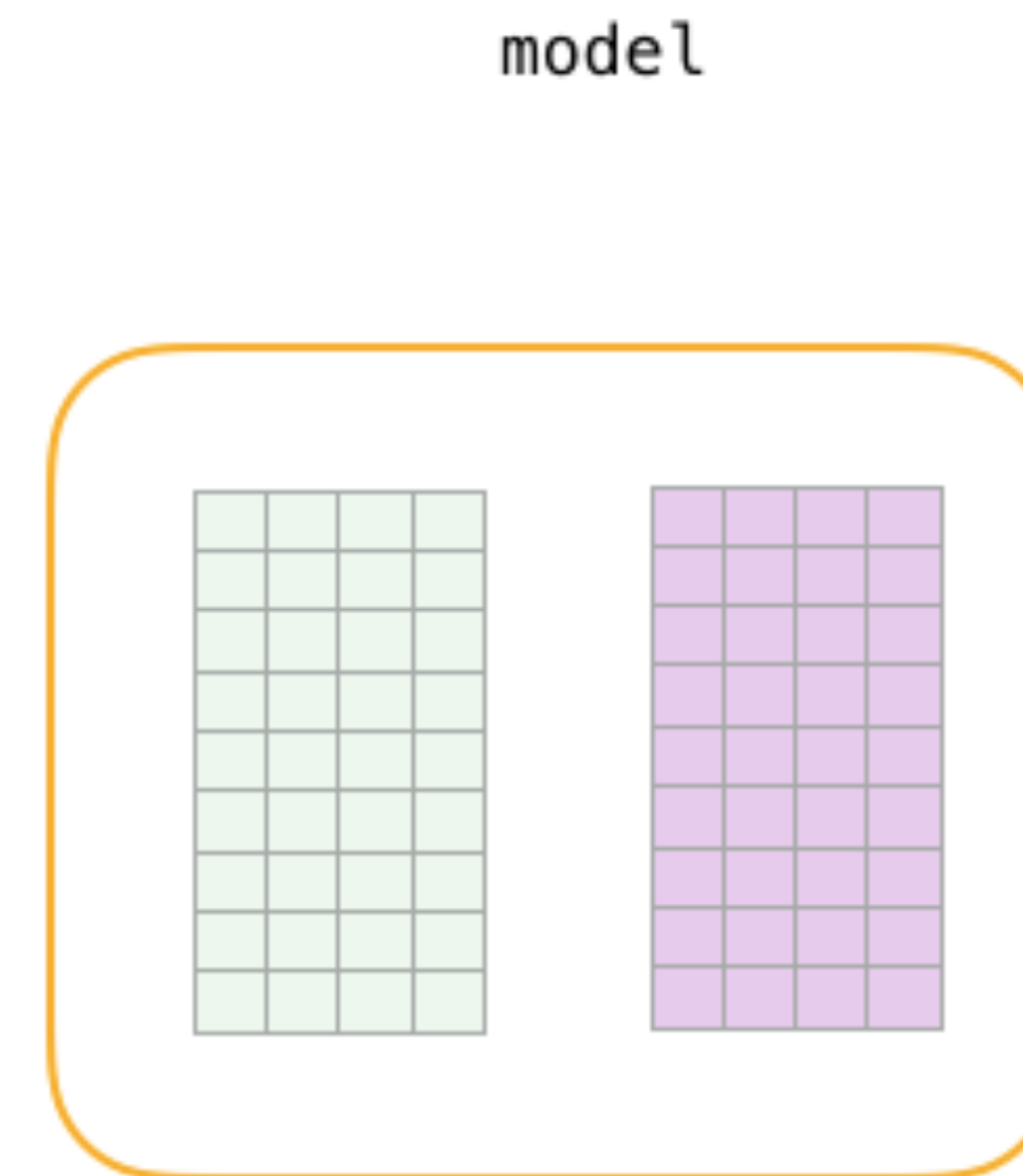
Update  
Model  
Parameters

## 48 Proceed to Next Batch

- Then we proceed to our next step (the next positive sample and its associated negative samples) and do the same process again.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...



# Machine Translation: Sequence to Sequence and Attention



# What is Machine Translation





# Commercial Machine Translation Systems

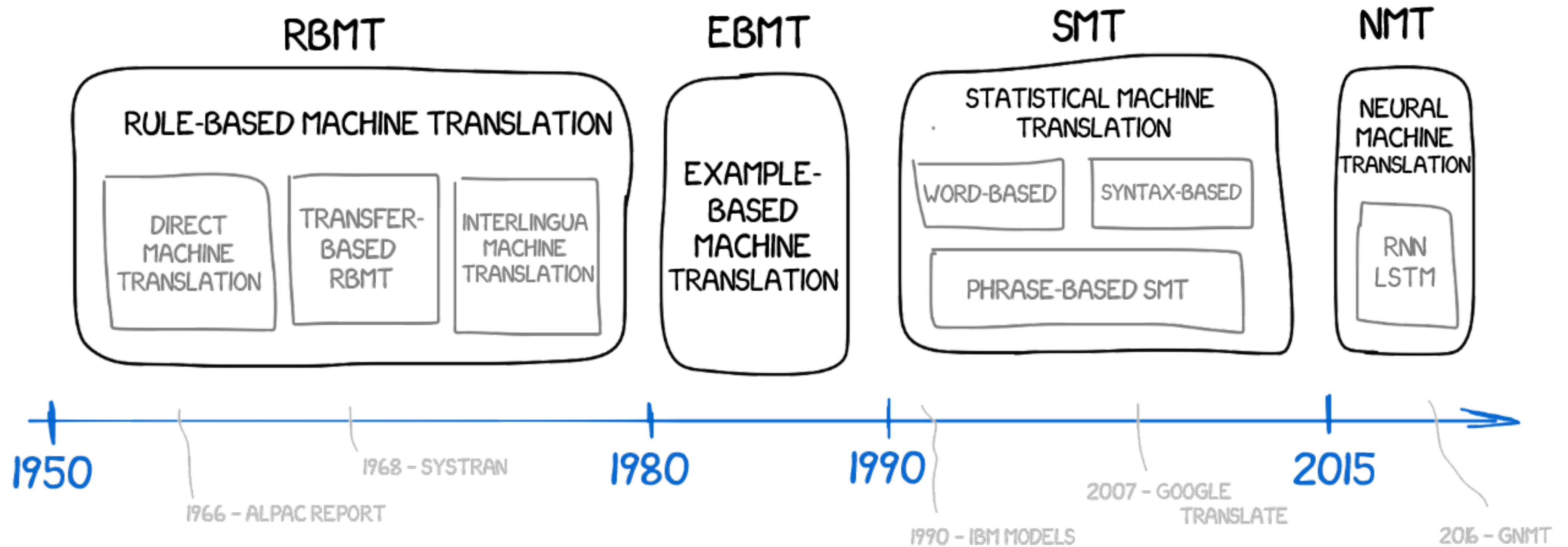
- Google Translate
- Yandex Translate
- Bing Microsoft Translator
- Baidu Fanyi
- DeepL Translator
- Wechat Translate
- .....

Google Translate



# History of Machine Translation

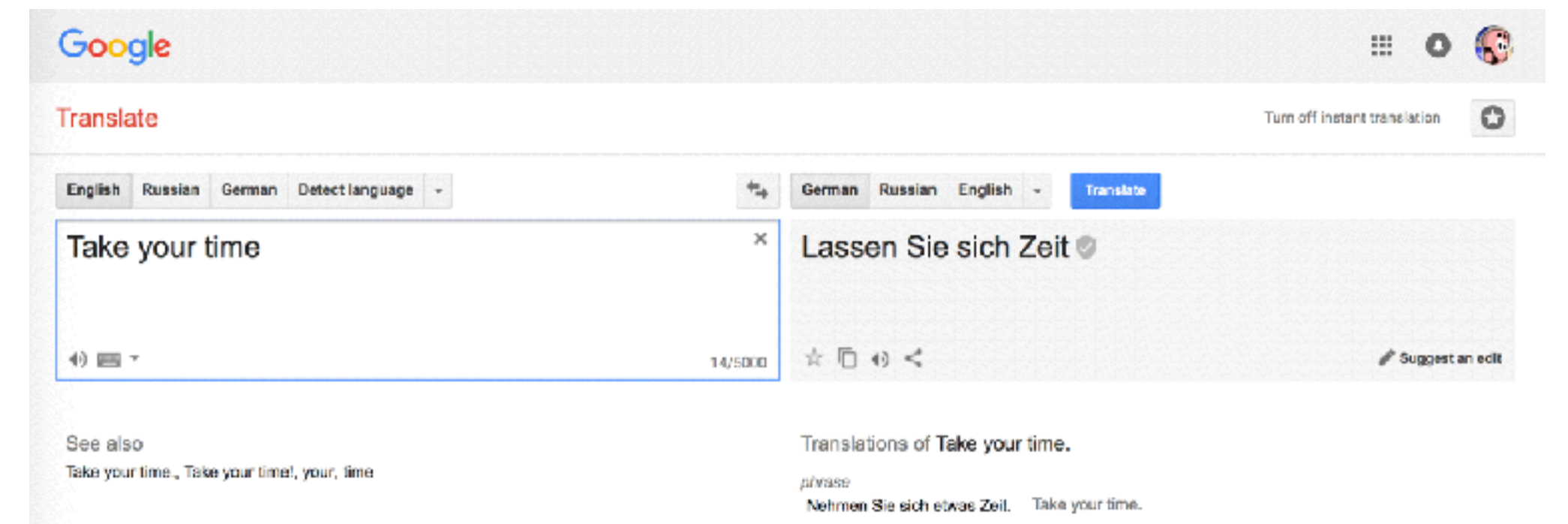
## A BRIEF HISTORY OF MACHINE TRANSLATION



# 53 2016: The Neural MT (NMT) Revolution

In November 2016, Google made a game-changing announcement announcing the launch of the Google Neural Machine Translation System (GNMT). The idea was similar to transferring style between photos such as programs like Prisma that can turn a photo into a painting imitating famous artists' style. If we can transfer the style to a photo, how about imposing a language to a source text?

The idea was to be able to translate while keeping the essence of the source text (just like the artist's style). The source text is encoded into a set of specific features by one neural network and then decoded back into text in the target language by another network. Both networks speak a different language and don't know about each other but they both can understand the set of features extracted from the source text. This is quite similar to the idea of Interlingua Machine Translation. In a few years, NMT surpassed every system that developed previously and with the implementation of deep learning, it was able to implement improvements without being taught to do so.





# Sequence to Sequence

Formally, in the machine translation task, we have an input sequence  $x_1, x_2, \dots, x_m$  and an output sequence  $y_1, y_2, \dots, y_n$  (note that their lengths can be different). Translation can be thought of as finding the target sequence that is the most probable given the input; formally, the target sequence that maximizes the conditional probability  $p(y|x)$ :  $y^* = \arg \max_y p(y|x)$ .

If you are bilingual and can translate between languages easily, you have an intuitive feeling of  $p(y|x)$  and can say something like "...well, this translation is kind of more natural for this sentence". But in machine translation, we learn a function  $p(y|x, \theta)$  with some parameters  $\theta$ , and then find its argmax for a given input:  $y' = \arg \max_y p(y|x, \theta)$ .



# 55 Sequence to Sequence

## Human Translation

$$y^* = \arg \max_y p(y|x)$$

The “probability” is intuitive and is given by a human translator’s expertise

## Machine Translation

$$y' = \arg \max_y p(y|x, \theta)$$

model parameters

Questions we need to answer

- **modeling**

How does the model for  $p(y|x, \theta)$  look like?

- **learning**

How to find  $\theta$ ?

- **search**

How to find the argmax?

## 56 Sequence to Sequence

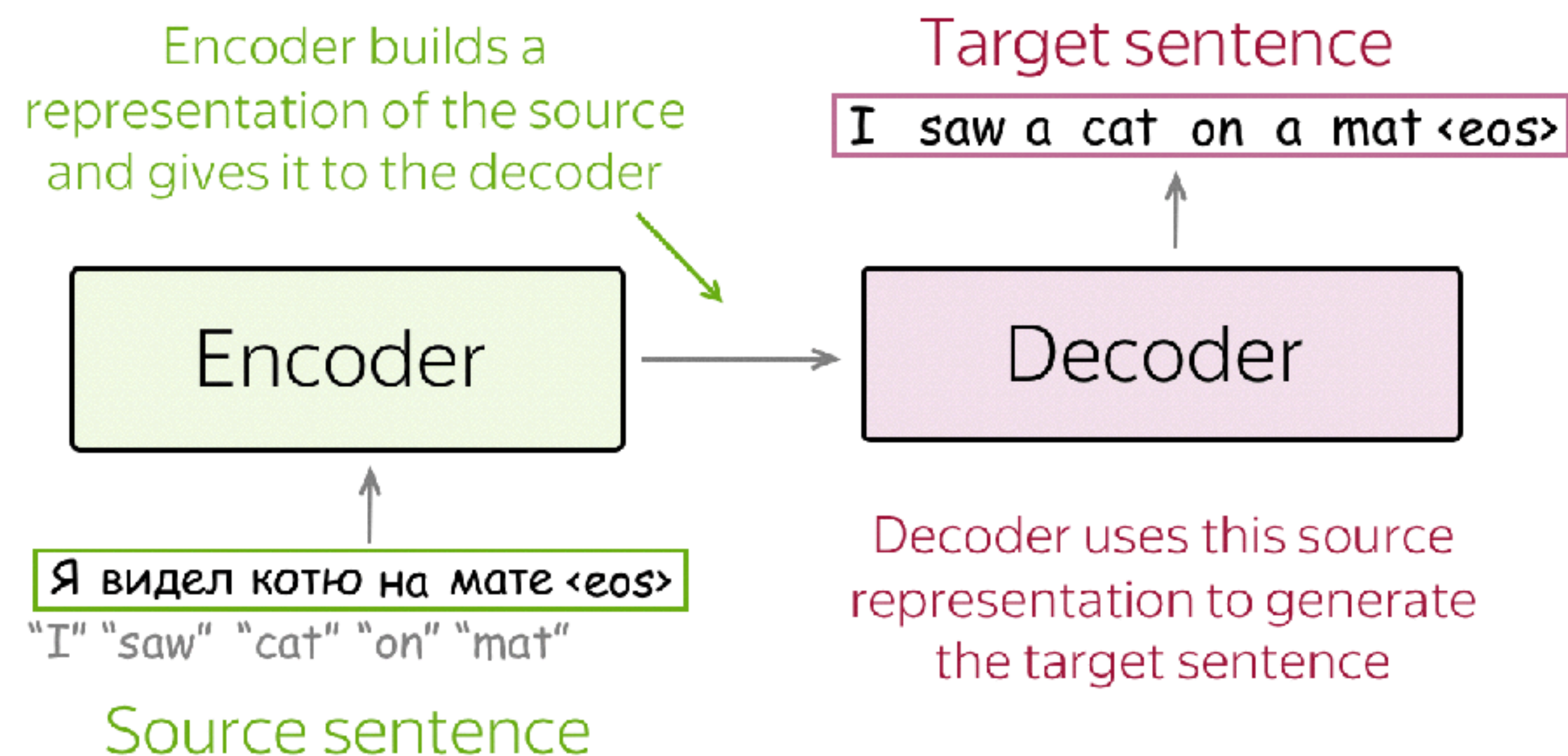
To define a machine translation system, we need to answer three questions:

- **modeling** - how does the model for  $p(y|x, \theta)$  look like?
- **learning** - how to find the parameters  $\theta$ ?
- **inference** - how to find the best  $y$ ?

## 57 Encoder-Decoder Framework

Encoder-decoder is the standard modeling paradigm for sequence-to-sequence tasks. This framework consists of two components:

- **encoder** - reads source sequence and produces its representation;
- **decoder** - uses source representation from the encoder to generate the target sequence.





# Conditional Language Models

In the [Language Modeling](#) lecture, we learned to estimate the probability  $p(\mathbf{y})$  of sequences of tokens  $\mathbf{y} = (y_1, y_2, \dots, y_n)$ . While language models estimate the unconditional probability  $p(\mathbf{y})$  of a sequence  $\mathbf{y}$ , sequence-to-sequence models need to estimate the conditional probability  $p(\mathbf{y}|\mathbf{x})$  of a sequence  $\mathbf{y}$  given a source  $\mathbf{x}$ . That's why sequence-to-sequence tasks can be modeled as **Conditional Language Models (CLM)** - they operate similarly to LMs, but additionally receive source information  $\mathbf{x}$ .

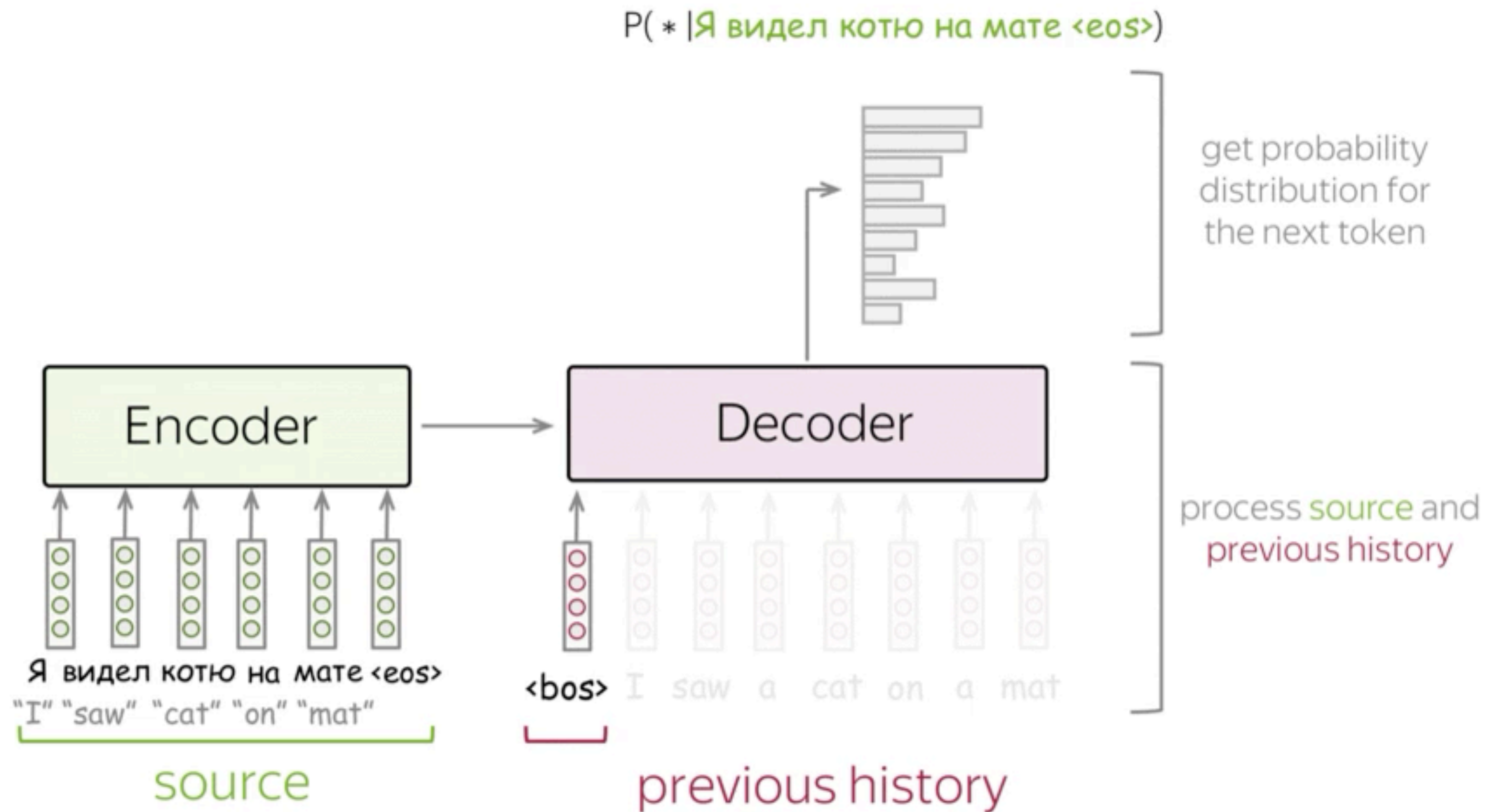
Language Models: 
$$P(y_1, y_2, \dots, y_n) = \prod_{t=1}^n p(y_t | y_{<t})$$

Conditional  
Language Models: 
$$P(y_1, y_2, \dots, y_n, | \mathbf{x}) = \prod_{t=1}^n p(y_t | y_{<t}, \mathbf{x})$$

condition on source  $\mathbf{x}$

Note that Conditional Language Modeling is something more than just a way to solve sequence-to-sequence tasks. In the most general sense,  **$\mathbf{x}$  can be something other than a sequence of tokens**. For example, in the Image Captioning task,  $\mathbf{x}$  is an image and  $\mathbf{y}$  is a description of this image.

# Conditional Language Models

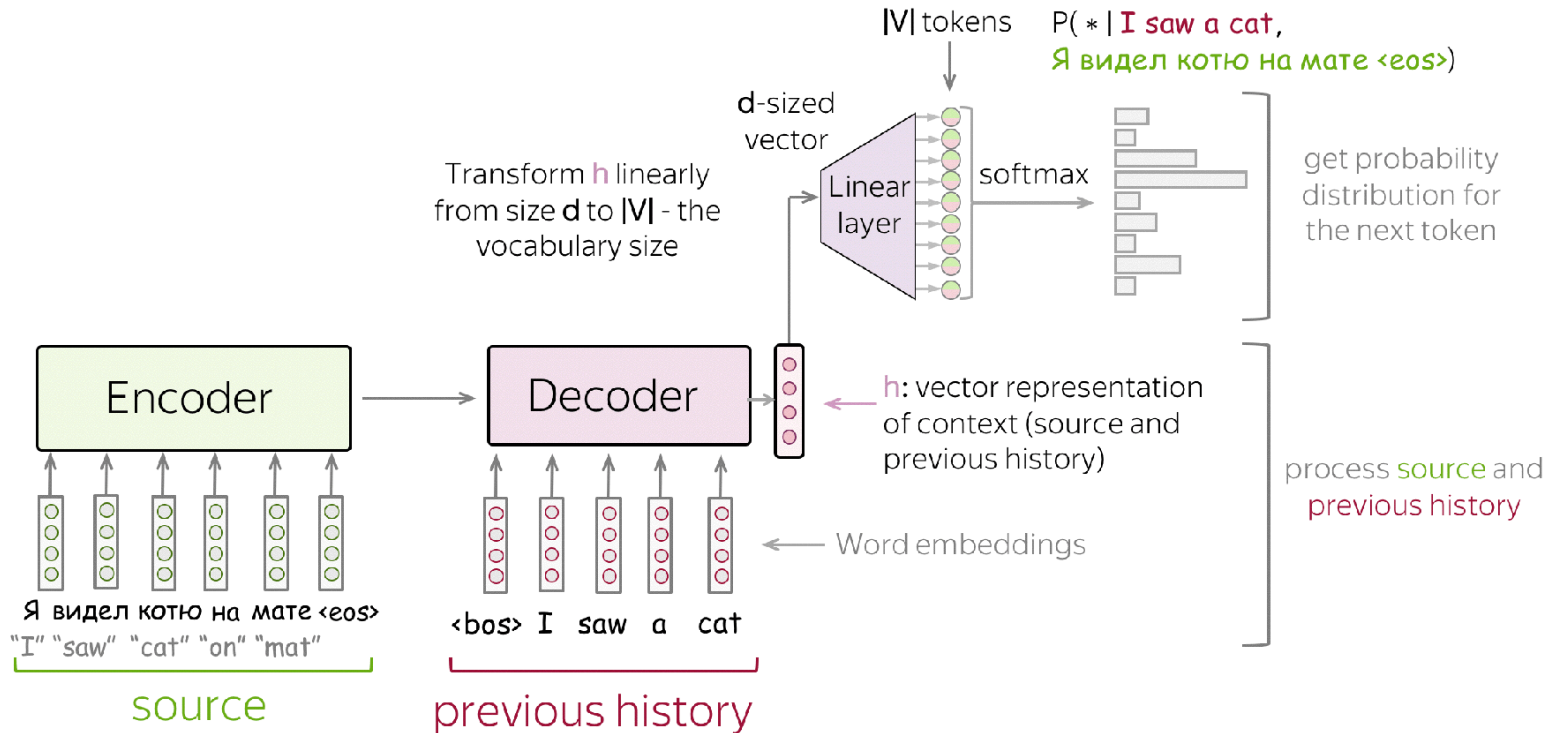




# Conditional Language Models

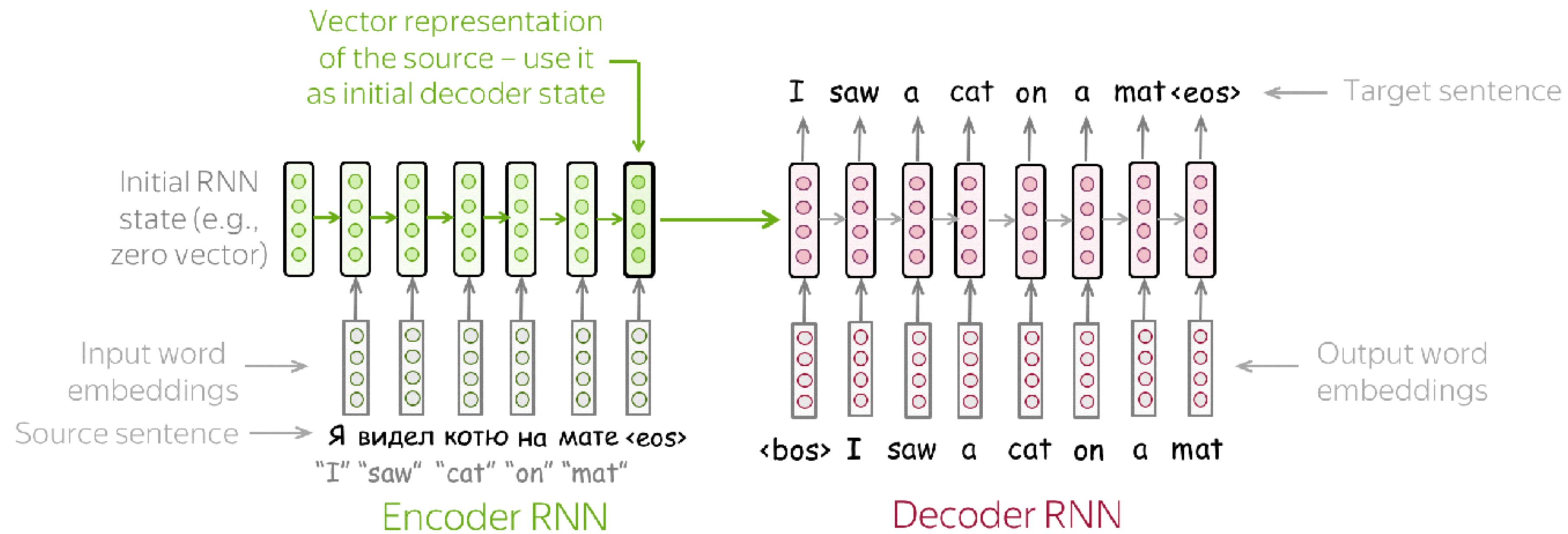
Since the only difference from LMs is the presence of source  $x$ , the modeling and training is very similar to language models. In particular, the high-level pipeline is as follows:

- feed source and previously generated target words into a network;
- get vector representation of context (both source and previous target) from the networks decoder;
- from this vector representation, predict a probability distribution for the next token.



## 62 Simple Model: Two RNNs for Encoder & Decoder

The simplest encoder-decoder model consists of two RNNs (LSTMs): one for the encoder and another for the decoder. Encoder RNN reads the source sentence, and the final state is used as the initial state of the decoder RNN. The hope is that the final encoder state "encodes" all information about the source, and the decoder can generate the target sentence based on this vector.

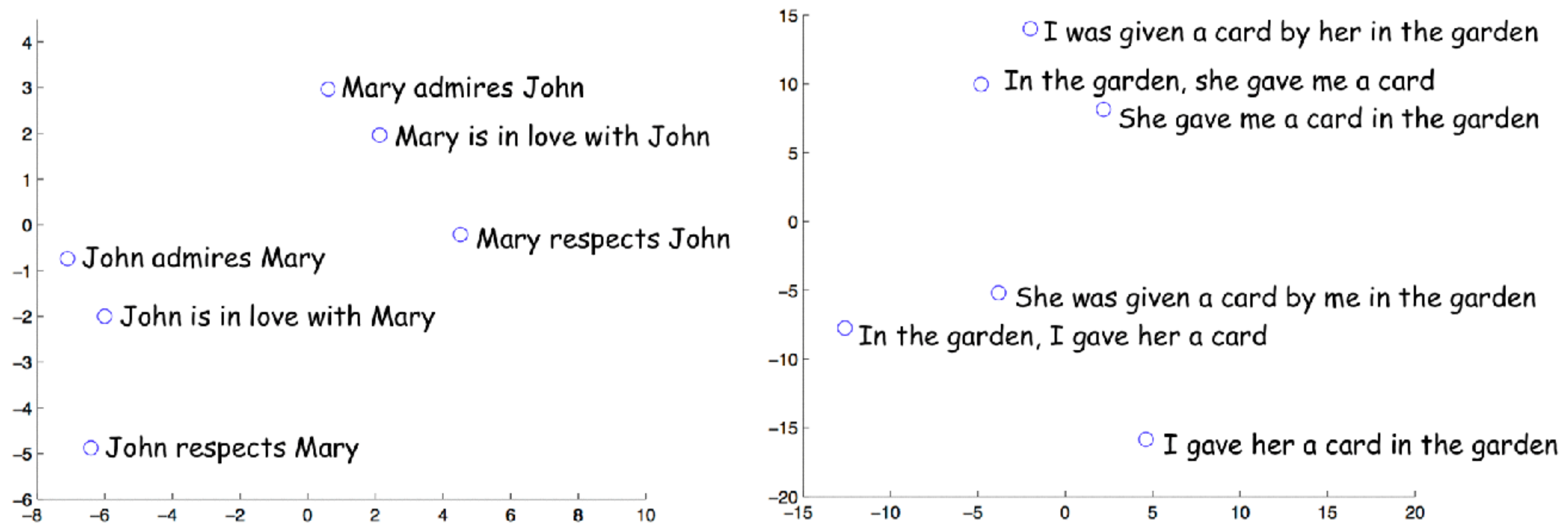




## 63 Simple Model: Two RNNs for Encoder & Decoder

This model can have different modifications: for example, the encoder and decoder can have **several layers**. Such a model with several layers was used, for example, in the paper [Sequence to Sequence Learning with Neural Networks](#) - one of the first attempts to solve sequence-to-sequence tasks using neural networks.

In the same paper, the authors looked at the last encoder state and visualized several examples - look below. Interestingly, **representations of sentences with similar meaning but different structure are close!**

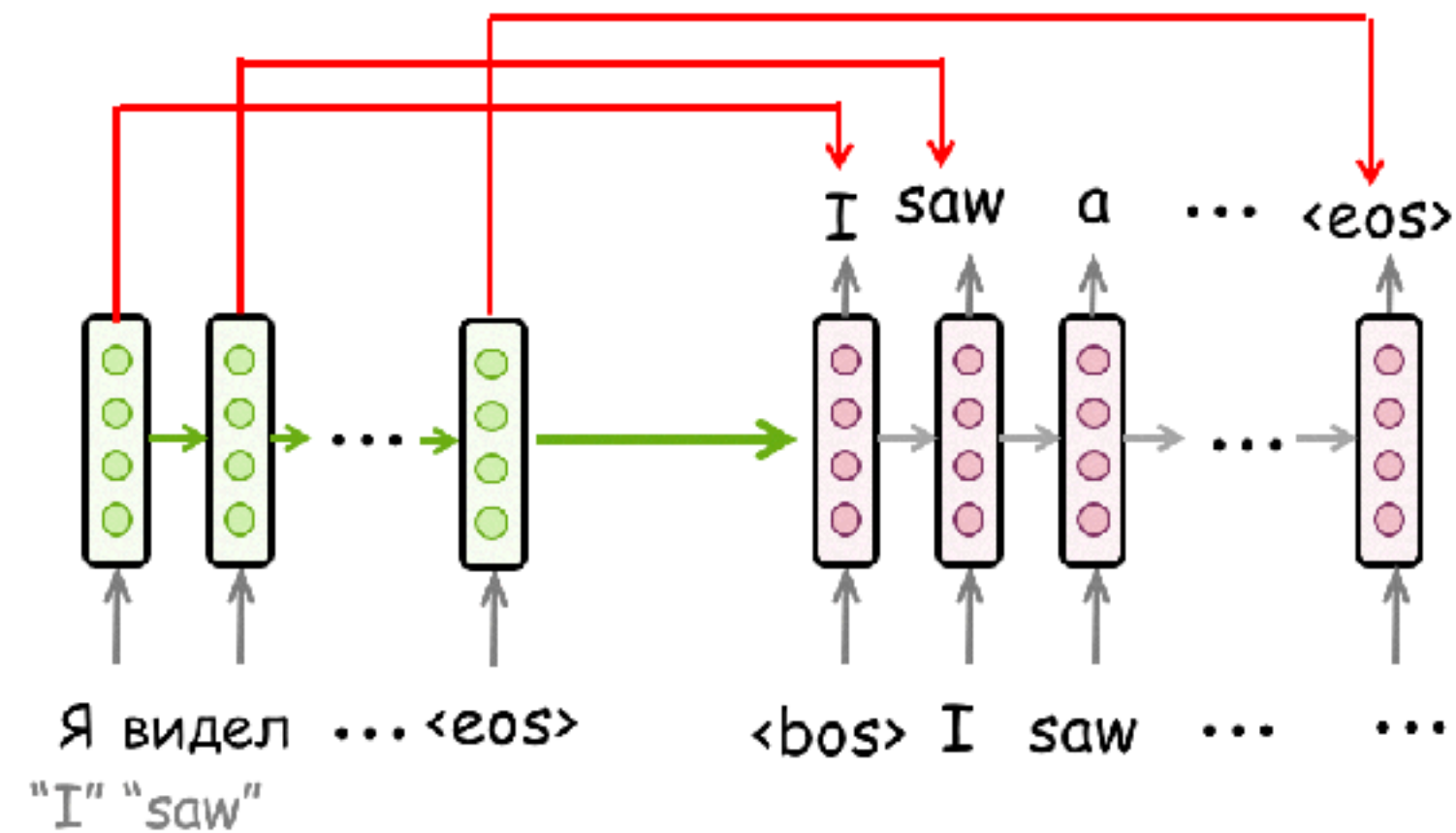


The examples are from the paper [Sequence to Sequence Learning with Neural Networks](#)

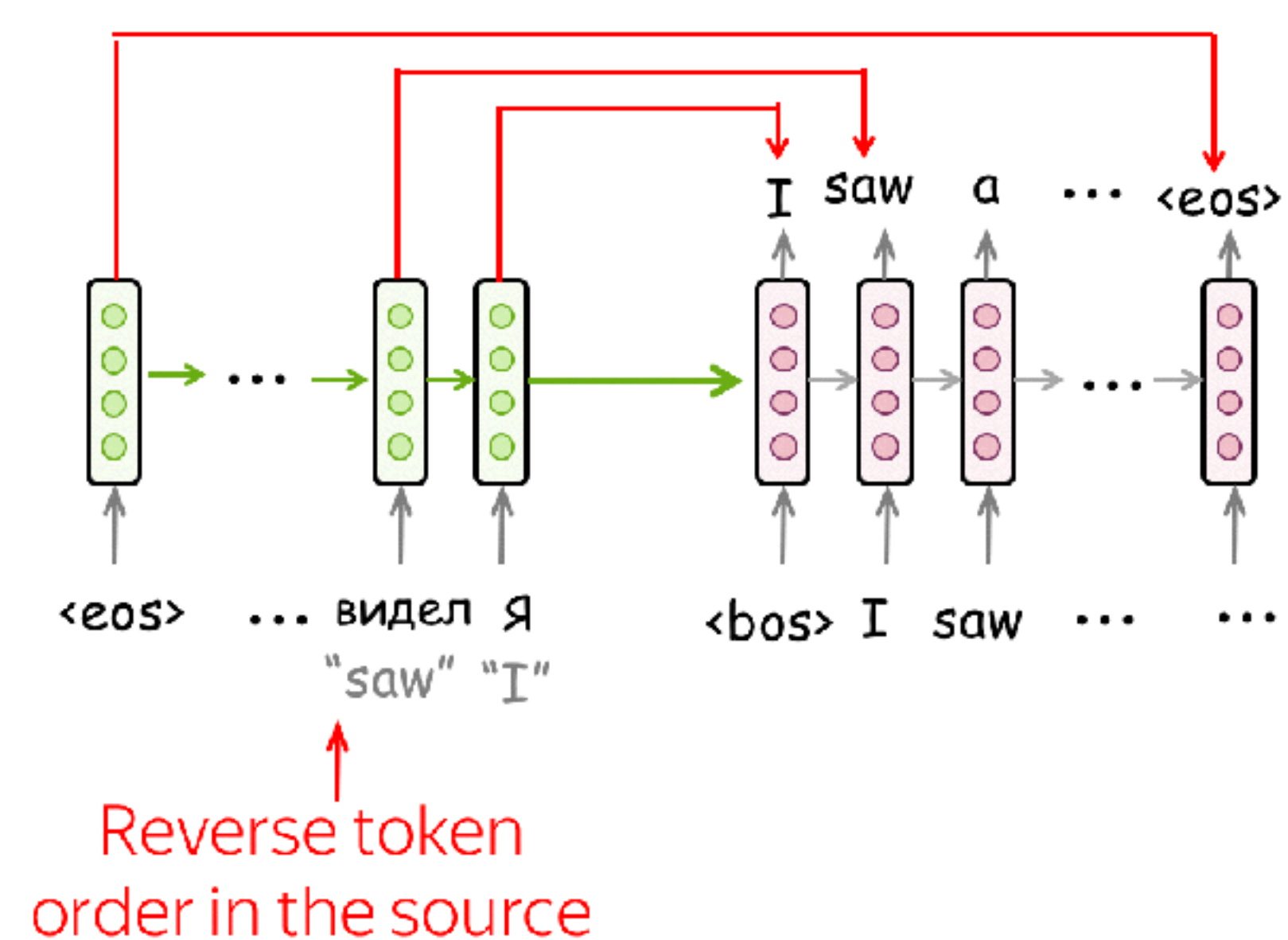
# 64 Training Trick: Reverse Order of Source Tokens

The paper [Sequence to Sequence Learning with Neural Networks](#) introduced an elegant trick to make simple LSTM seq2seq models work better: reverse the order of the source tokens (but not the target). After that, a model will have many short-term connections: the latest source tokens it sees are the most relevant for the beginning of the target.

Before: all dependencies are long-term



After: many short-term dependencies





## Training: The Cross-Entropy Loss

Similarly to neural LMs, neural seq2seq models are trained to predict probability distributions of the next token given previous context (source and previous target tokens). Intuitively, at each step we maximize the probability a model assigns to the correct token.

Formally, let's assume we have a training instance with the source  $\mathbf{x} = (x_1, \dots, x_m)$  and the target  $\mathbf{y} = (y_1, \dots, y_n)$ . Then at the timestep  $t$ , a model predicts a probability distribution  $p^{(t)} = p(*|y_1, \dots, y_{t-1}, x_1, \dots, x_m)$ . The target at this step is  $p^* = \text{one-hot}(y_t)$ , i.e., we want a model to assign probability 1 to the correct token,  $y_t$ , and zero to the rest.

The standard loss function is the **cross-entropy loss**. Cross-entropy loss for the target distribution  $p^*$  and the predicted distribution  $p$  is

$$\text{Loss}(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{|V|} p_i^* \log(p_i).$$

Since only one of  $p_i^*$  is non-zero (for the correct token  $y_t$ ), we will get

$$\text{Loss}(p^*, p) = -\log(p_{y_t}) = -\log(p(y_t|y_{<t}, \mathbf{x})).$$

# Training: The Cross-Entropy Loss

At each step, we maximize the probability a model assigns to the correct token. Look at the illustration for a single timestep.

Source sequence:

Я видел котю на мате <eos>  
"I" "saw" "cat" "on" "mat"

Target sequence:

I saw a cat on a mat <eos>

← one training example

← one step for this example

previous tokens      we want the model to predict this

Model prediction:  $p(* | \text{I saw a, Я ... } \langle \text{eos} \rangle)$

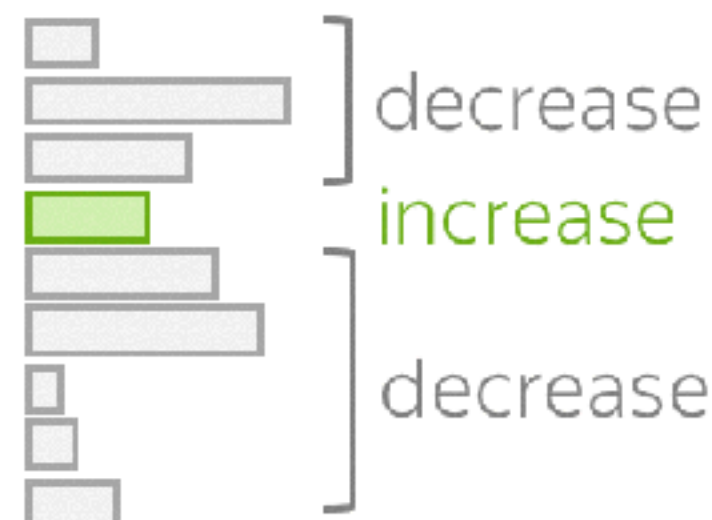


Target



← cat →

Loss =  $-\log(p(\text{cat})) \rightarrow \min$

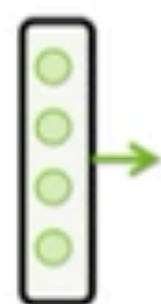


decrease  
increase  
decrease

## 67 Training: The Cross-Entropy Loss

For the whole example, the loss will be  $-\sum_{t=1}^n \log(p(y_t | y_{<t}, x))$ . Look at the illustration of the training process (the illustration is for the RNN model, but the model can be different).

Encoder: read source



we are here  
Source: Я видел котю на мате <eos>  
"I" "saw" "cat" "on" "mat"

Target: I saw a cat on a mat <eos>

## 68 Inference

Now when we understand how a model can look like and how to train this model, let's think how to generate a translation using this model. We model the probability of a sentence as follows:

$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t | y_{<t}, x) \quad \text{How to find the argmax?}$$

Now the main question is: how to find the argmax?

Note that **we can not find the exact solution**. The total number of hypotheses we need to check is  $|V|^n$ , which is not feasible in practice. Therefore, we will find an approximate solution.



# Inference: Greedy Decoding

- **Greedy Decoding:** At each step, pick the most probable token

The straightforward decoding strategy is greedy - at each step, generate a token with the highest probability. This can be a good baseline, but this method is inherently flawed: the best token at the current step does not necessarily lead to the best sequence.

$$\arg \max_y \prod_{t=1}^n p(y_t | y_{<t}, x) \neq \prod_{t=1}^n \arg \max_{y_t} p(y_t | y_{<t}, x)$$



## 70 Inference: Beam Search

- **Beam Search:** Keep track of several most probably hypotheses

Instead, let's keep several hypotheses. At each step, we will be continuing each of the current hypotheses and pick top-N of them. This is called **beam search**.

<bos>

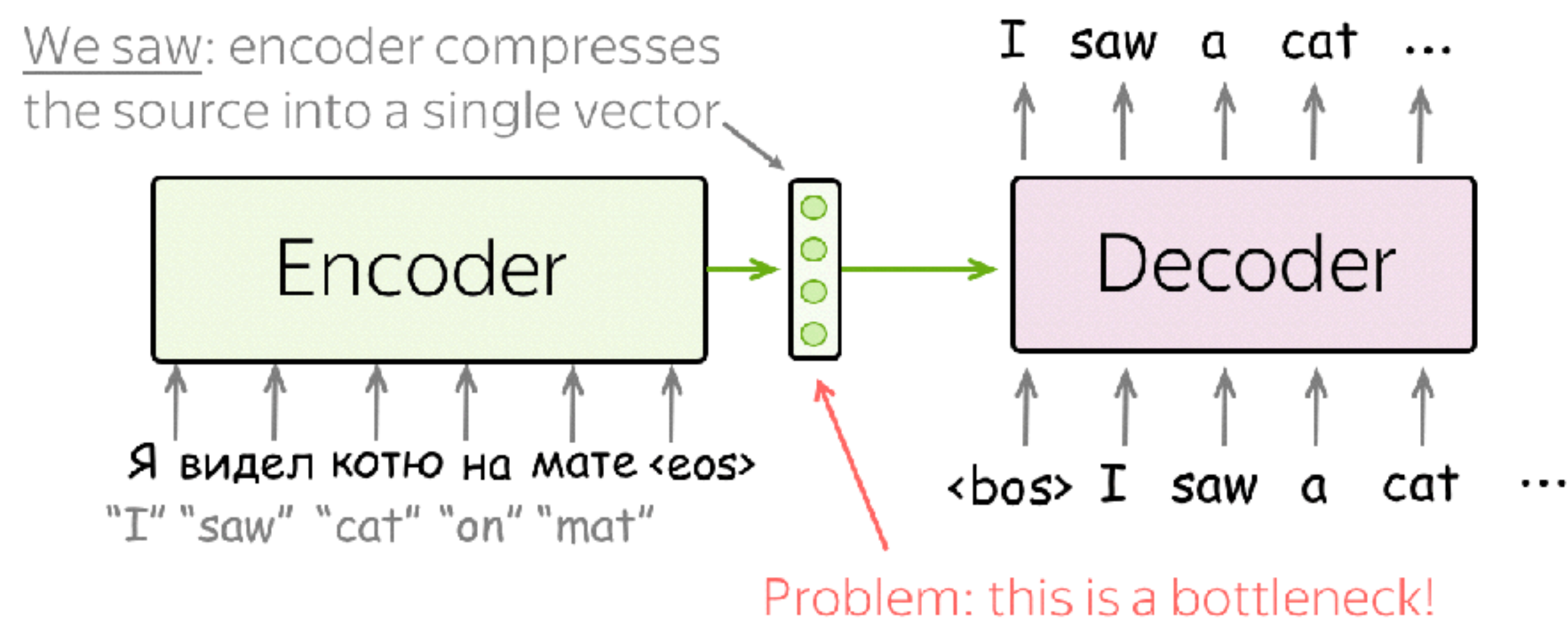
Start with the begin of sentence token or with an empty sequence

Usually, the beam size is 4-10. Increasing beam size is computationally inefficient and, what is more important, leads to worse quality.

# 71 The Problem of Fixed Encoder Representation

Problem: Fixed source representation is suboptimal: (i) for the encoder, it is hard to compress the sentence; (ii) for the decoder, different information may be relevant at different steps.

In the models we looked at so far, the encoder compressed the whole source sentence into a single vector. This can be very hard - the number of possible meanings of source is infinite. When the encoder is forced to put all information into a single vector, it is likely to forget something.



Not only it is hard for the encoder to put all information into a single vector - this is also hard for the decoder. The decoder sees only one representation of source. However, at each generation step, different parts of source can be more useful than others. But in the current setting, the decoder has to extract relevant information from the same fixed representation - hardly an easy thing to do.

## 72 Attention: A High-Level View

Attention was introduced in the paper [Neural Machine Translation by Jointly Learning to Align and Translate](#) to address the fixed representation problem.

**Attention: At different steps, let a model "focus" on different parts of the input.**

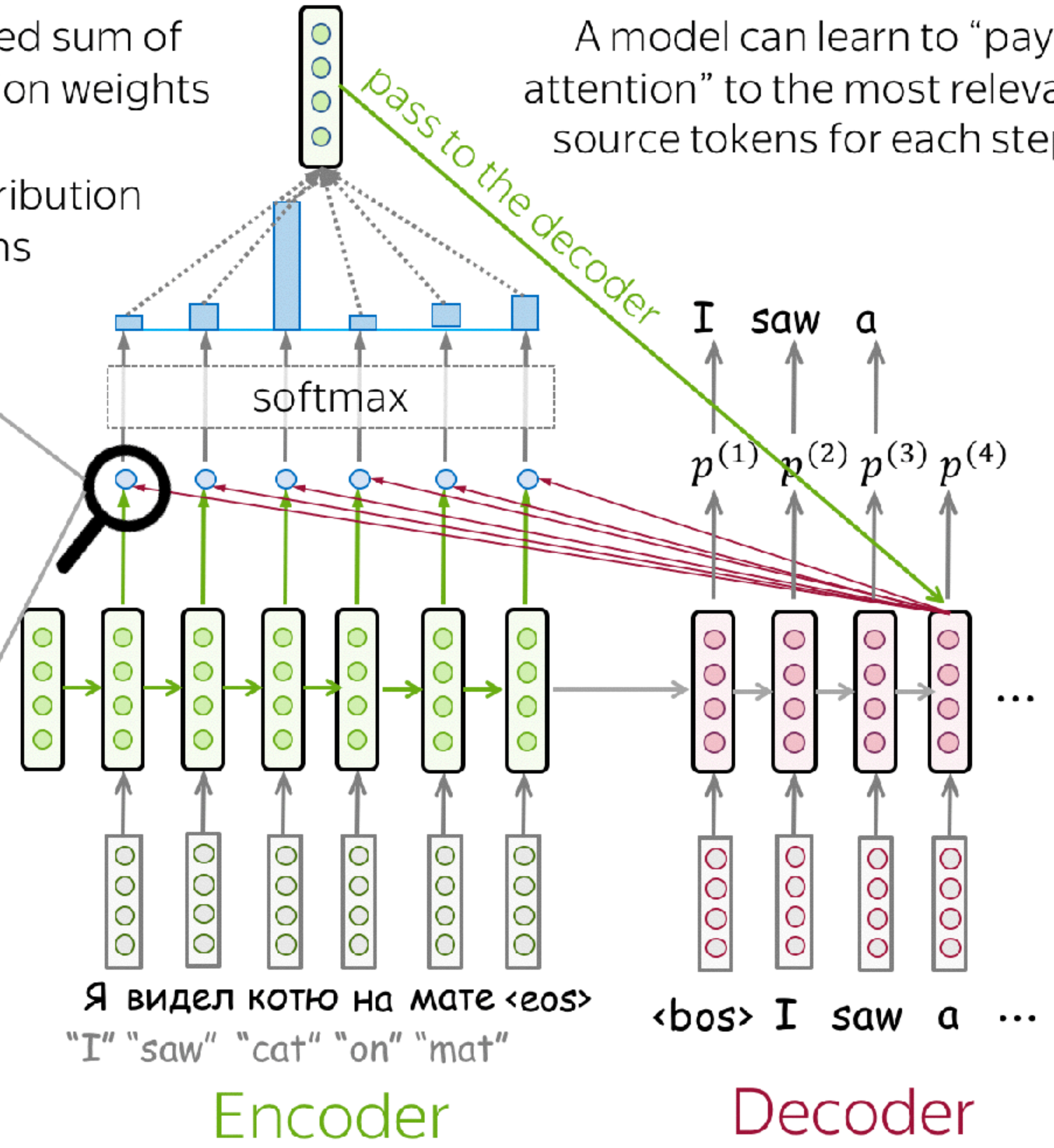
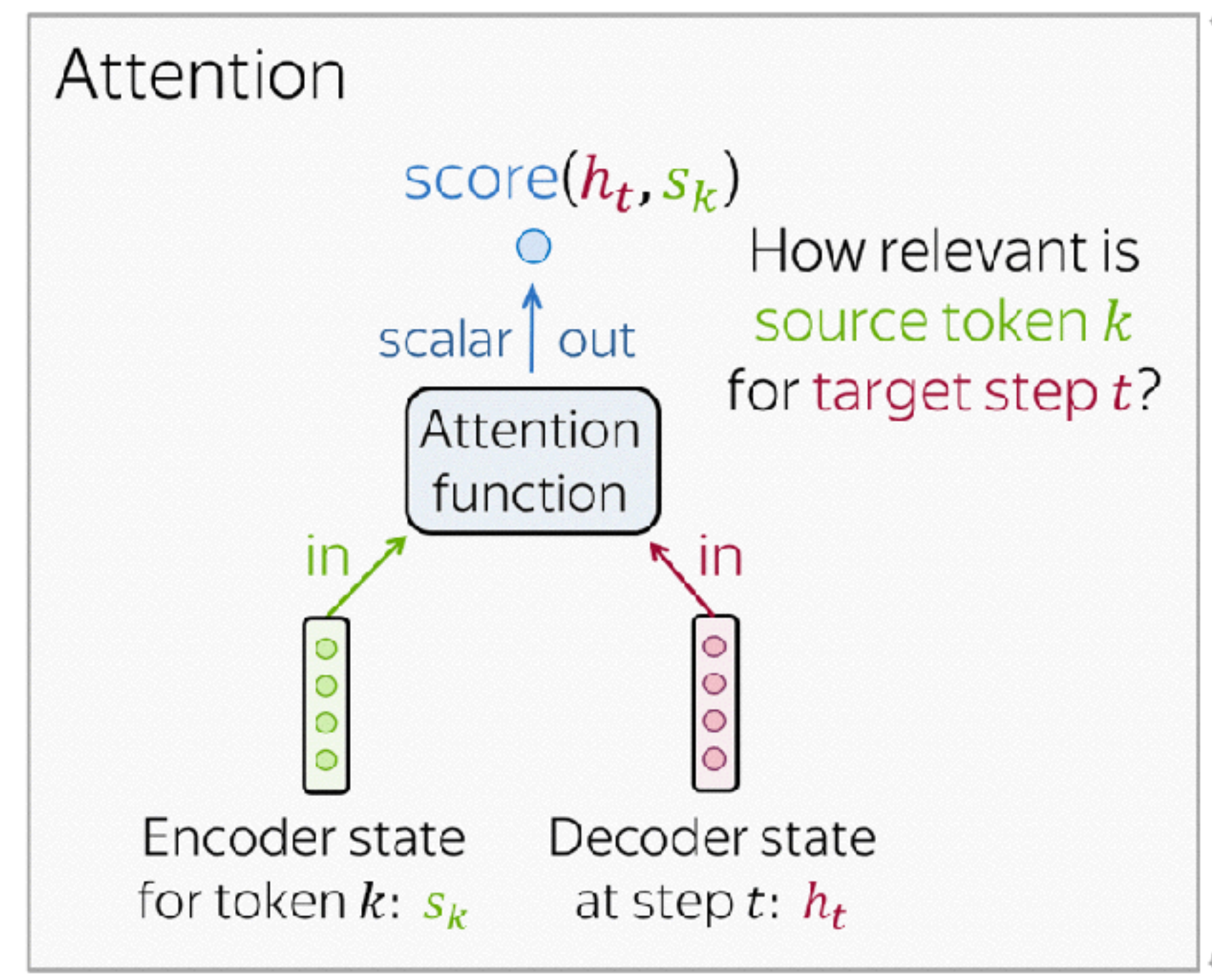
An attention mechanism is a part of a neural network. At each decoder step, it decides which source parts are more important. In this setting, the encoder does not have to compress the whole source into a single vector - it gives representations for all source tokens (for example, all RNN states instead of the last one).



Attention output: weighted sum of encoder states with attention weights

A model can learn to “pay attention” to the most relevant source tokens for each step

Attention weights: distribution over source tokens

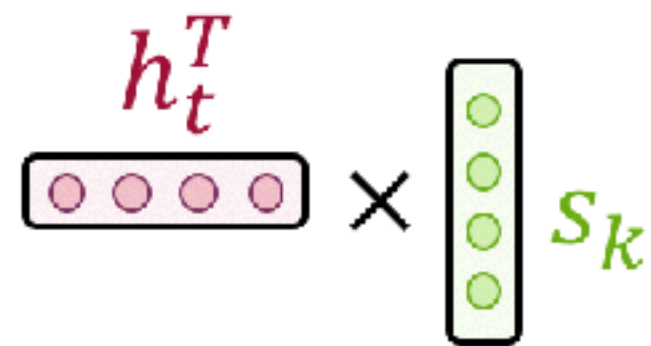


## 74 How Attention Works

The most popular ways to compute attention scores are:

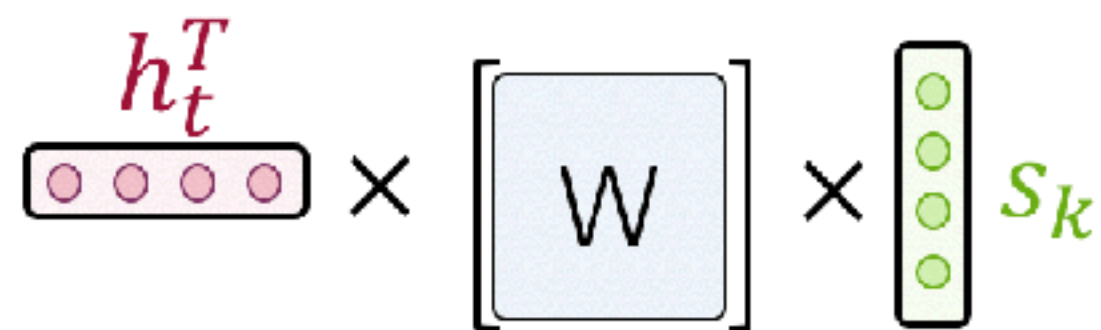
- dot-product - the simplest method;
- bilinear function (aka "Luong attention") - used in the paper [Effective Approaches to Attention-based Neural Machine Translation](#);
- multi-layer perceptron (aka "Bahdanau attention") - the method proposed in the [original paper](#).

Dot-product



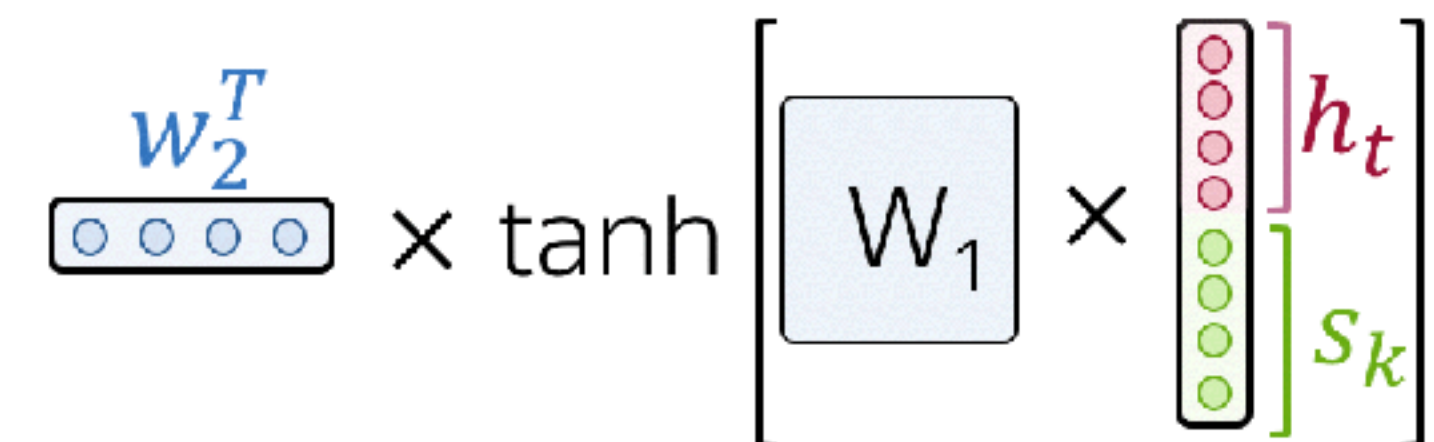
$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear



$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$



## 75 How Attention Works

At each decoder step, attention

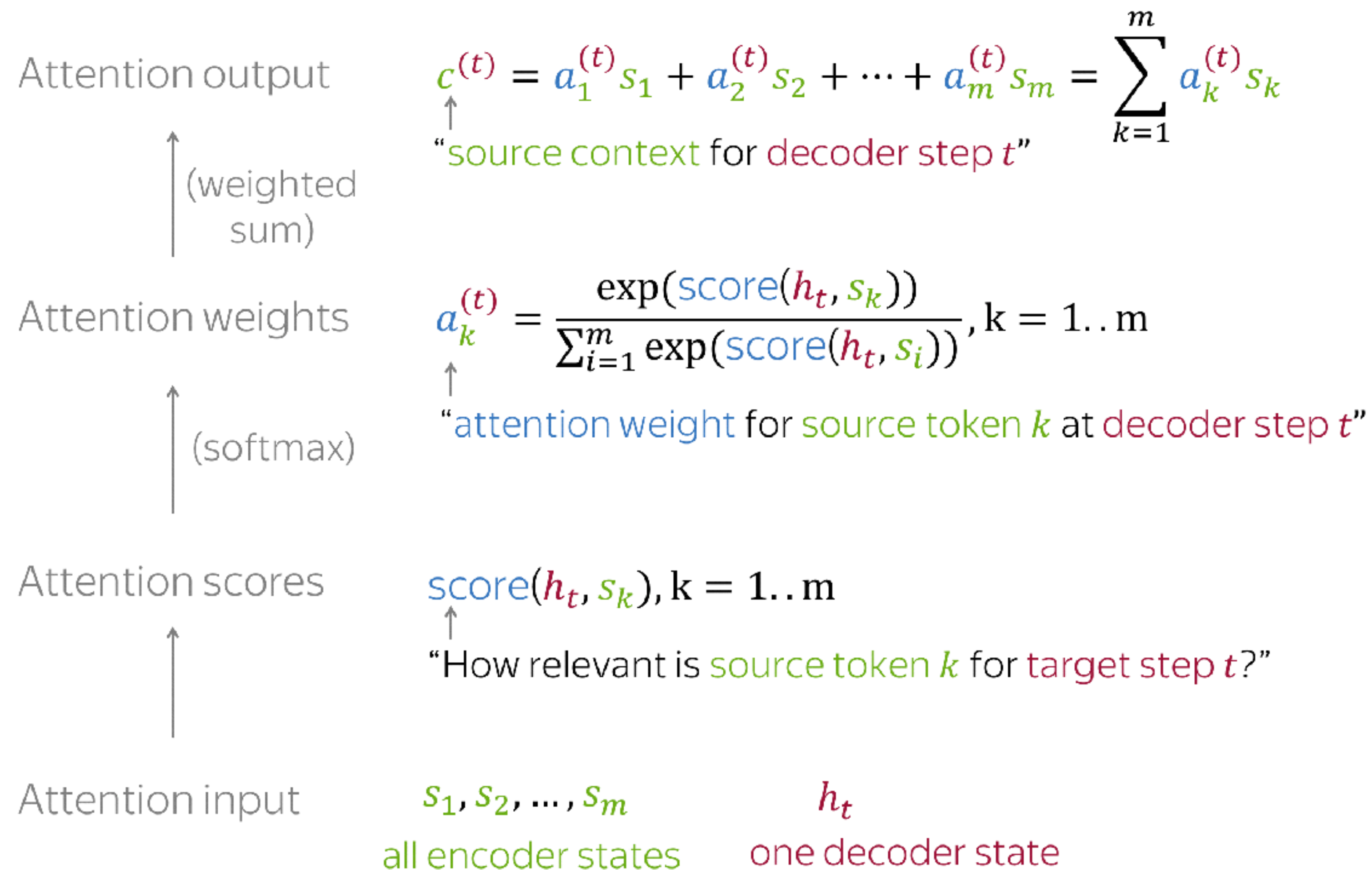
- receives **attention input**: a decoder state  $h_t$  and all encoder states  $s_1, s_2, \dots, s_m$ ;
- computes **attention scores**

For each encoder state  $s_k$ , attention computes its "relevance" for this decoder state  $h_t$ . Formally, it applies an attention function which receives one decoder state and one encoder state and returns a scalar value  $score(h_t, s_k)$ ;

- computes **attention weights**: a probability distribution - softmax applied to attention scores;
- computes **attention output**: the weighted sum of encoder states with attention weights.

# 76 How Attention Works

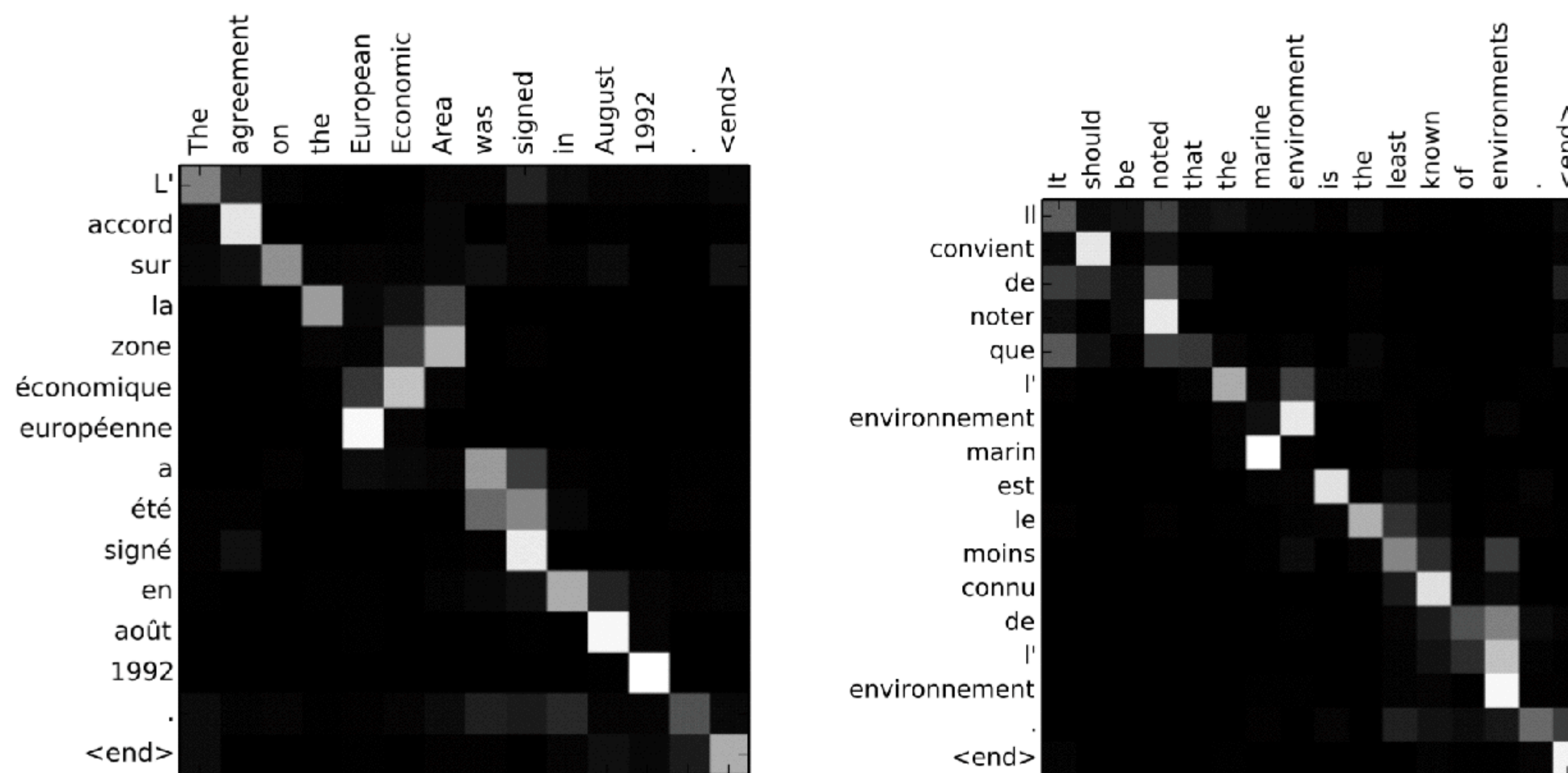
The general computation scheme is shown below.



**Note: Everything is differentiable - learned end-to-end!**

## 77 Attention Learns (Nearly) Alignments

Remember the motivation for attention? At different steps, the decoder may need to focus different source tokens, the ones which are more relevant at this step. Let's look at attention weights - which source words does the decoder use?



The examples are from the paper [Neural Machine Translation by Jointly Learning to Align and Translate](https://arxiv.org/abs/1410.3682).



# Transformer





# What is Transformer

- A model introduced in the paper “**Attention is All You Need**” in 2017.
- Based **solely on attention** mechanisms (i.e., no recurrence or convolutions).
- Higher translation quality, faster to train.

	Seq2seq without attention	Seq2seq with attention	Transformer
processing within <b>encoder</b>	RNN/CNN	RNN/CNN	attention
processing within <b>decoder</b>	RNN/CNN	RNN/CNN	attention
<b>decoder-encoder</b> interaction	static fixed-sized vector	attention	attention



# What We Just Saw

## Encoder

Who is doing:

- all source tokens

What they are doing:

- look at each other
  - update representations
- repeat  
N times

## Decoder

Who is doing:

- target token at the current step

What they are doing:

- looks at previous target tokens
  - looks at source representations
  - update representation
- repeat  
N times

# Why Such Design

- RNN won't understand what "bank" means until they read the whole sentence.
- Transformer's encoder tokens interact with each other all at once.

I arrived at the **bank** after crossing the ... ..street? ...river?

What does **bank** mean in this sentence?



I've no idea: let's wait until I read the end

RNNs

$O(N)$  steps to process a sentence with length  $N$



I don't need to wait - I see all words at once!

Transformer

Constant number of steps to process any sentence



# How to Implement

## 84 Self-Attention: the "Look at Each Other" Part

- Self-attention is one of the key components of the model.
- The difference between attention and self-attention is that self-attention operates between representations of the same nature: e.g., all encoder states in some layer.

Decoder-encoder attention is looking

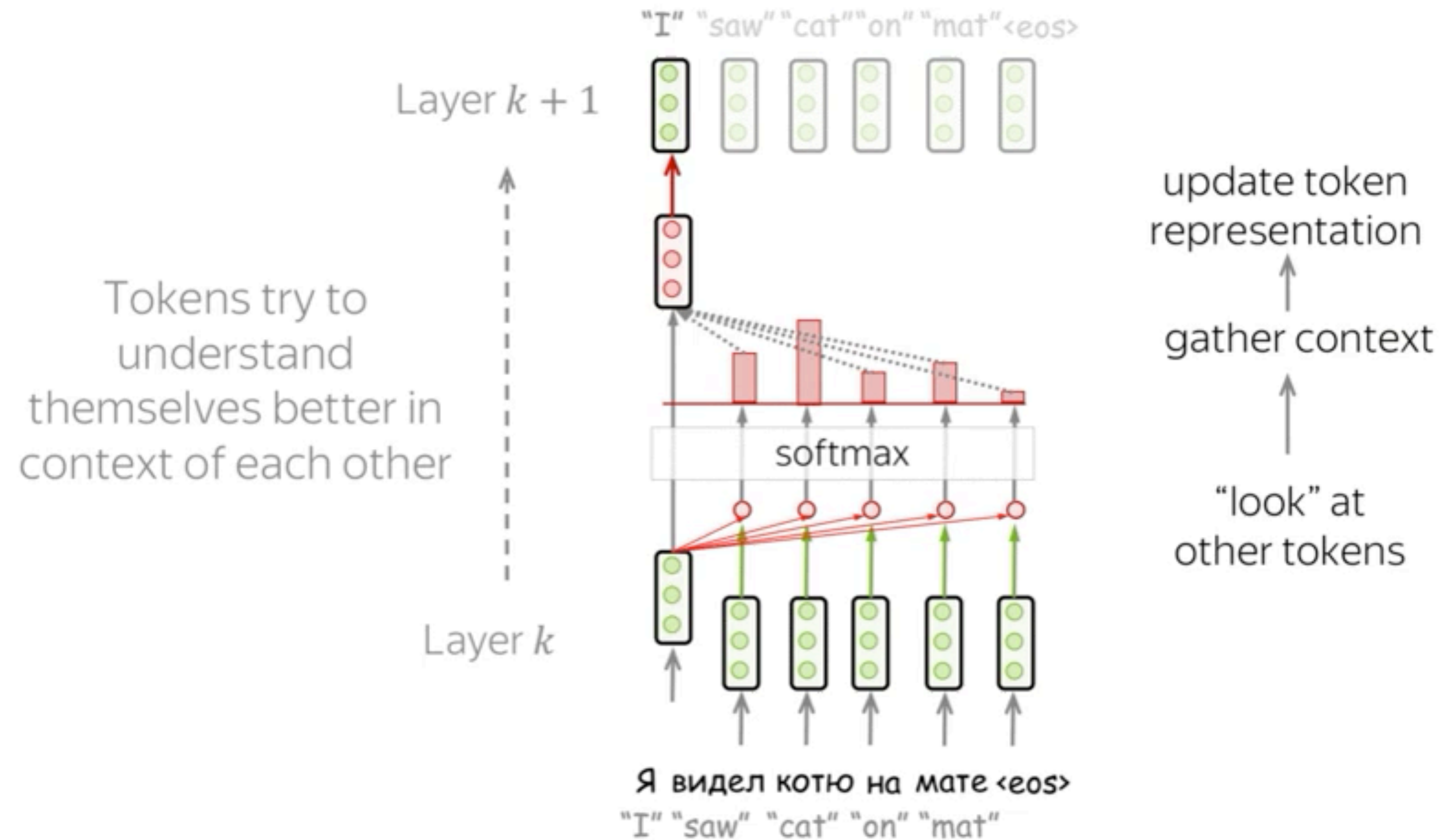
- **from:** one current decoder state
- **at:** all encoder states

Self-attention is looking

- **from:** each state from a set of states
- **at:** all other states in the same set

# 85 Self-Attention: the "Look at Each Other" Part

- **Self-attention** is the part of the model where **tokens interact with each other**.
- Each token "looks" at other tokens in the sentence with an attention mechanism, gathers context, and updates the previous representation of "self".
- Note that in practice, this happens **in parallel**.



## 86 Query, Key, and Value in Self-Attention

- ⦿ Each input token in **self-attention** receives three representations corresponding to the roles it can play:
  - **query** - asking for information;
  - **key** - saying that it has some information;
  - **value** - giving the information.
- ⦿ The **query** is used when a token looks at others - it's **seeking the information** to understand itself better.
- ⦿ The **key** is **responding to a query's request**: it is used to **compute attention weights**.
- ⦿ The **value** is used to **compute attention output**: it gives information to the tokens which "say" they need it (i.e. assigned large weights to this token).



Each vector receives three representations (“roles”)

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

**Query:** vector from which the attention is looking

“Hey there, do you have this information?”

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

**Key:** vector at which the query looks to compute weights

“Hi, I have this information – give me a large weight!”

$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

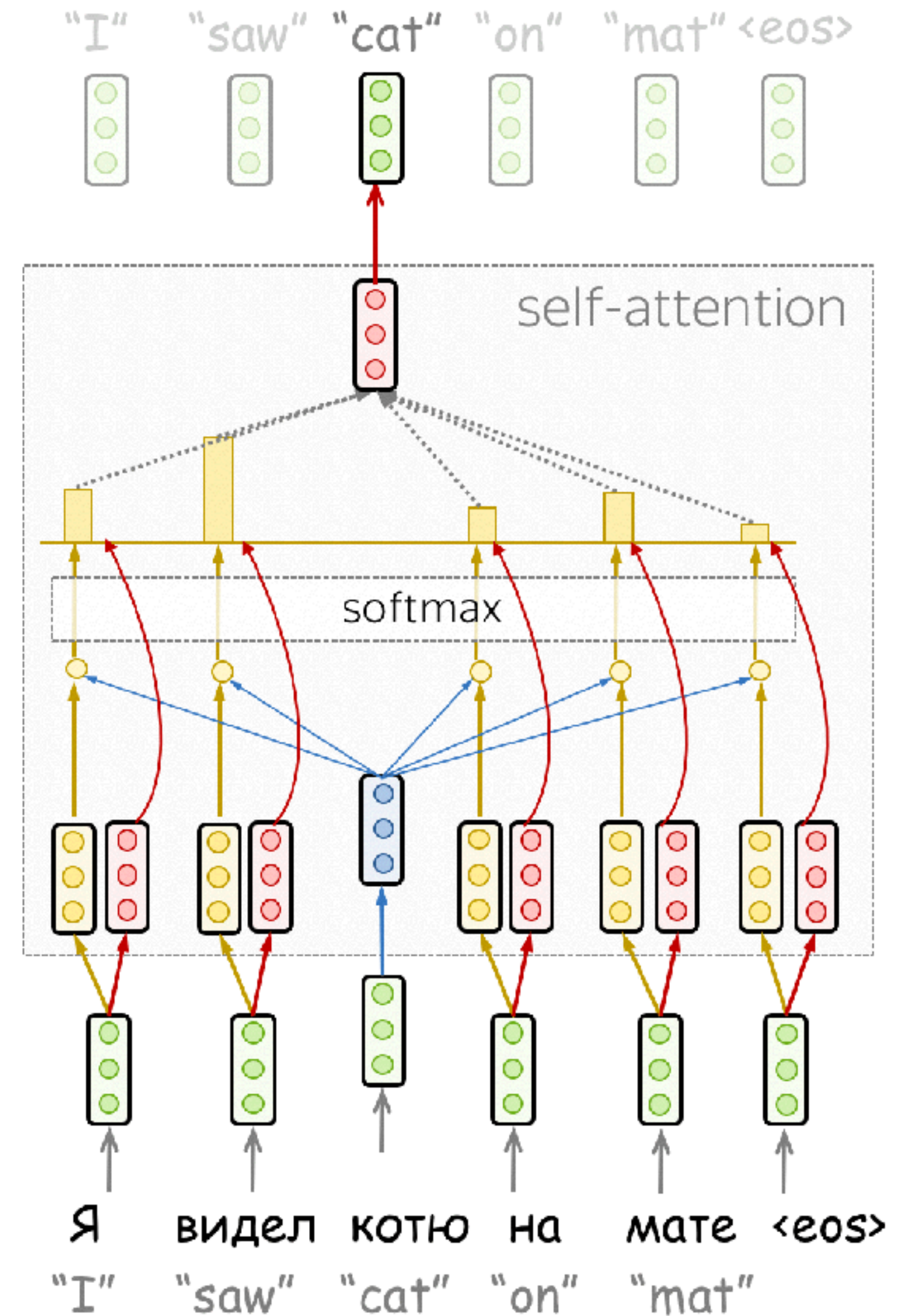
**Value:** their weighted sum is attention output

“Here’s the information I have!”

$$Attention(q, k, v) = \overbrace{\text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from to

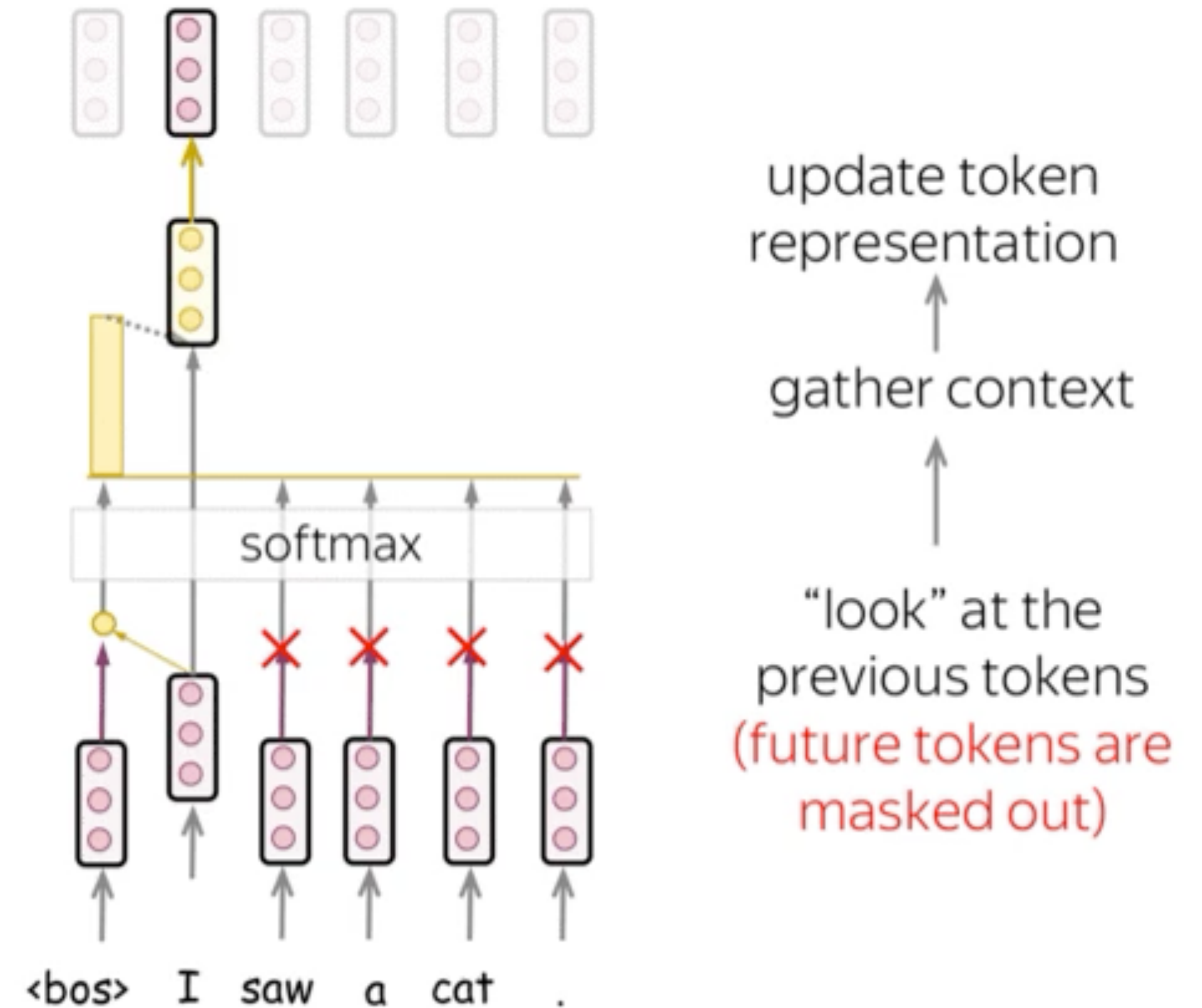
vector dimensionality of K, V



# Masked Self-Attention

## "Don't Look Ahead" for the Decoder

- In the **decoder**, there's also a **self-attention** mechanism: it is the one performing the "**look at the previous tokens**" function.
- In the decoder, self-attention is a bit **different from the one in the encoder**. While the encoder receives all tokens at once and the tokens can look at all tokens in the input sentence, in the decoder, we generate one token at a time: **during generation, we don't know which tokens we'll generate in future**.
- To forbid the decoder to look ahead, the model uses **masked self-attention**: future tokens are masked out. Look at the illustration.





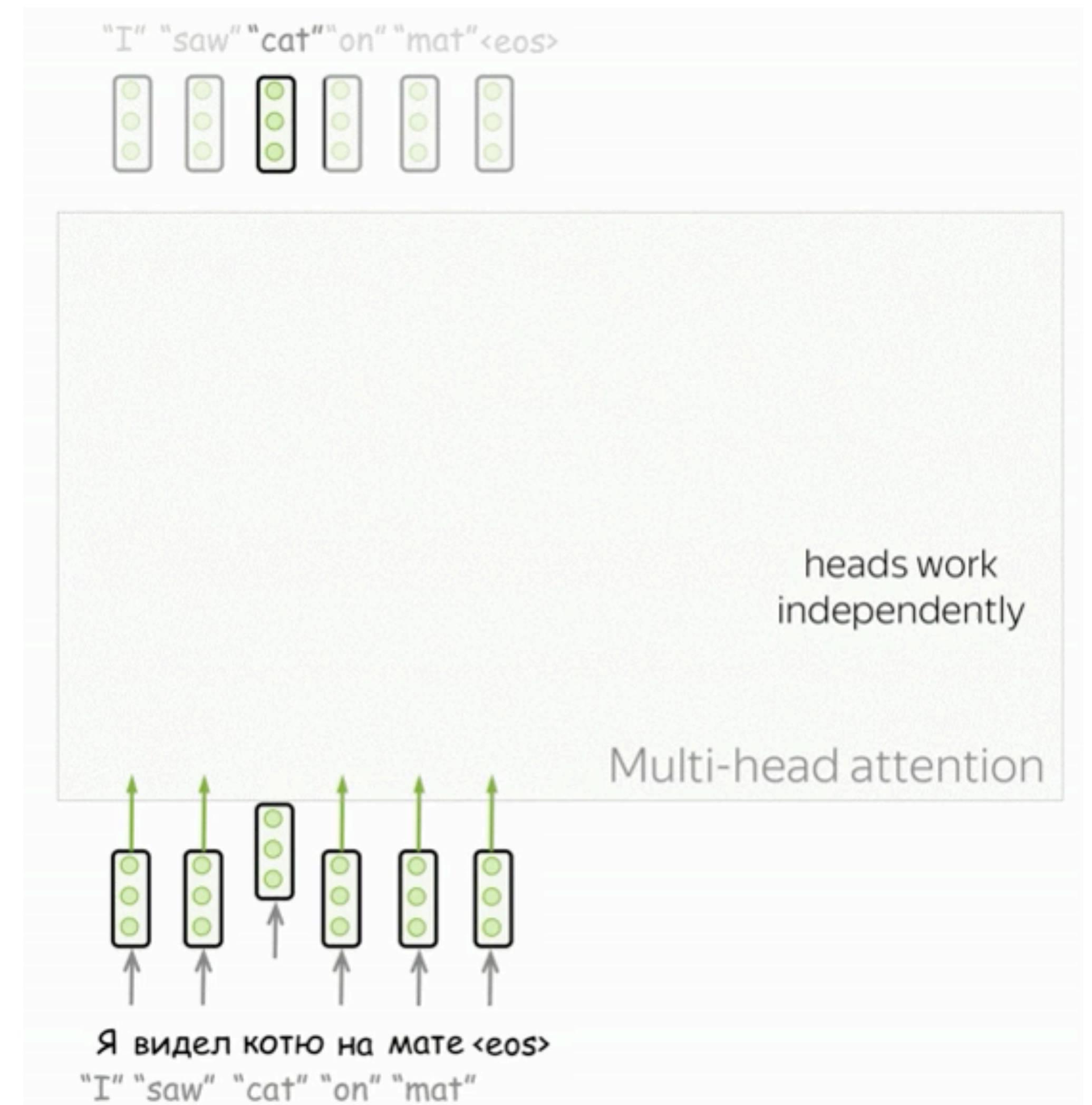
## 89 But How Can The Decoder Look Ahead?

- During generation, it can't - we don't know what comes next.
- But in training, we use reference translations (which we know). Therefore, in training, we feed the whole target sentence to the decoder - without masks, the tokens would "see future", and this is not what we want.
- This is done for computational efficiency: **the Transformer does not have a recurrence, so all tokens can be processed at once**. This is one of the reasons it has become so popular for machine translation - **it's much faster to train** than the once dominant recurrent models. For recurrent models, one training step requires  $O(\text{len}(\text{source}) + \text{len}(\text{target}))$  steps, but for Transformer, it's  $O(1)$ , i.e. constant.

# Multi-Head Attention

## Independently Focus on Different Things

- Usually, understanding the role of a word in a sentence requires understanding how it is related to different parts of the sentence.
- This is important not only in processing source sentence but also in generating target. For example, in some languages, subjects define verb inflection (e.g., gender agreement), verbs define the case of their objects, and many more. What I'm trying to say is: **each word is part of many relations.**
- Therefore, we have to **let the model focus on different things**: this is the motivation behind Multi-Head Attention. **Instead of having one attention mechanism, multi-head attention has several "heads" which work independently.**

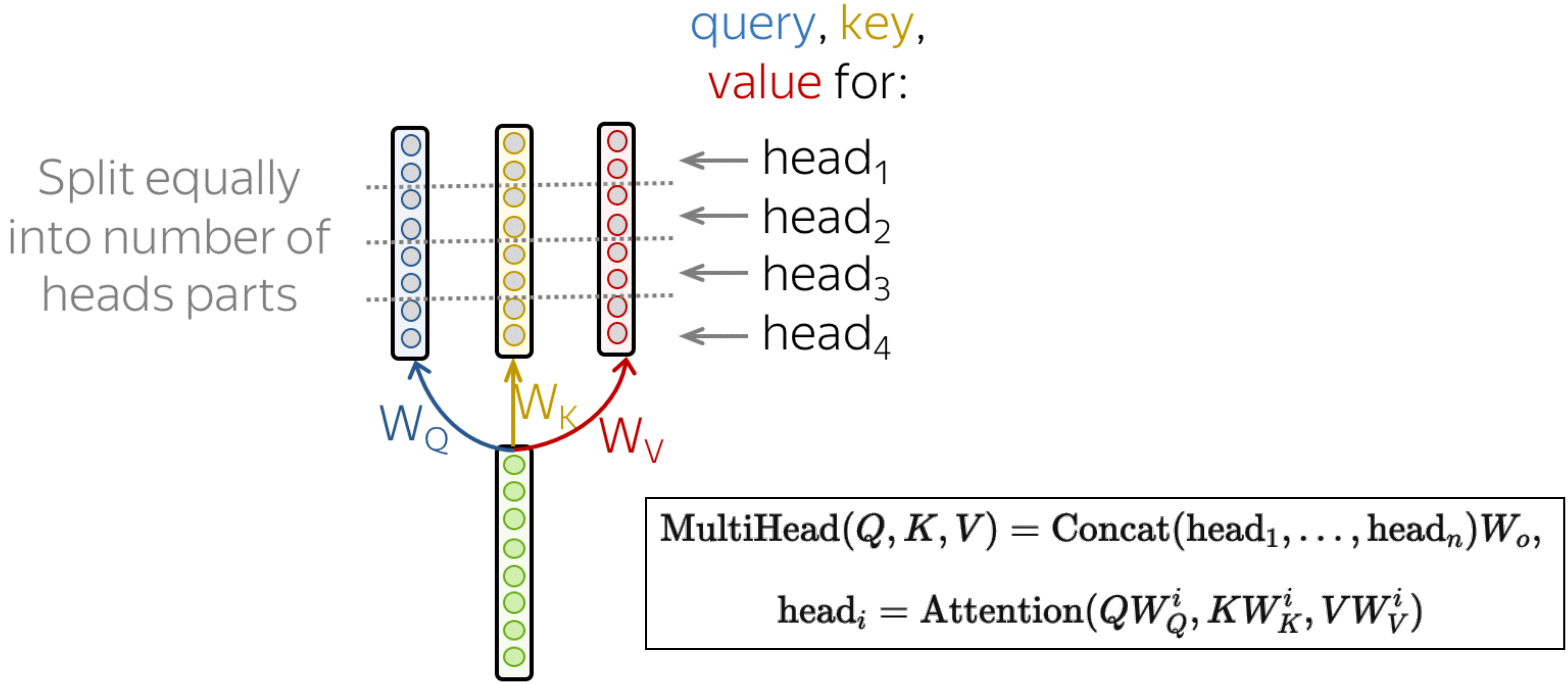




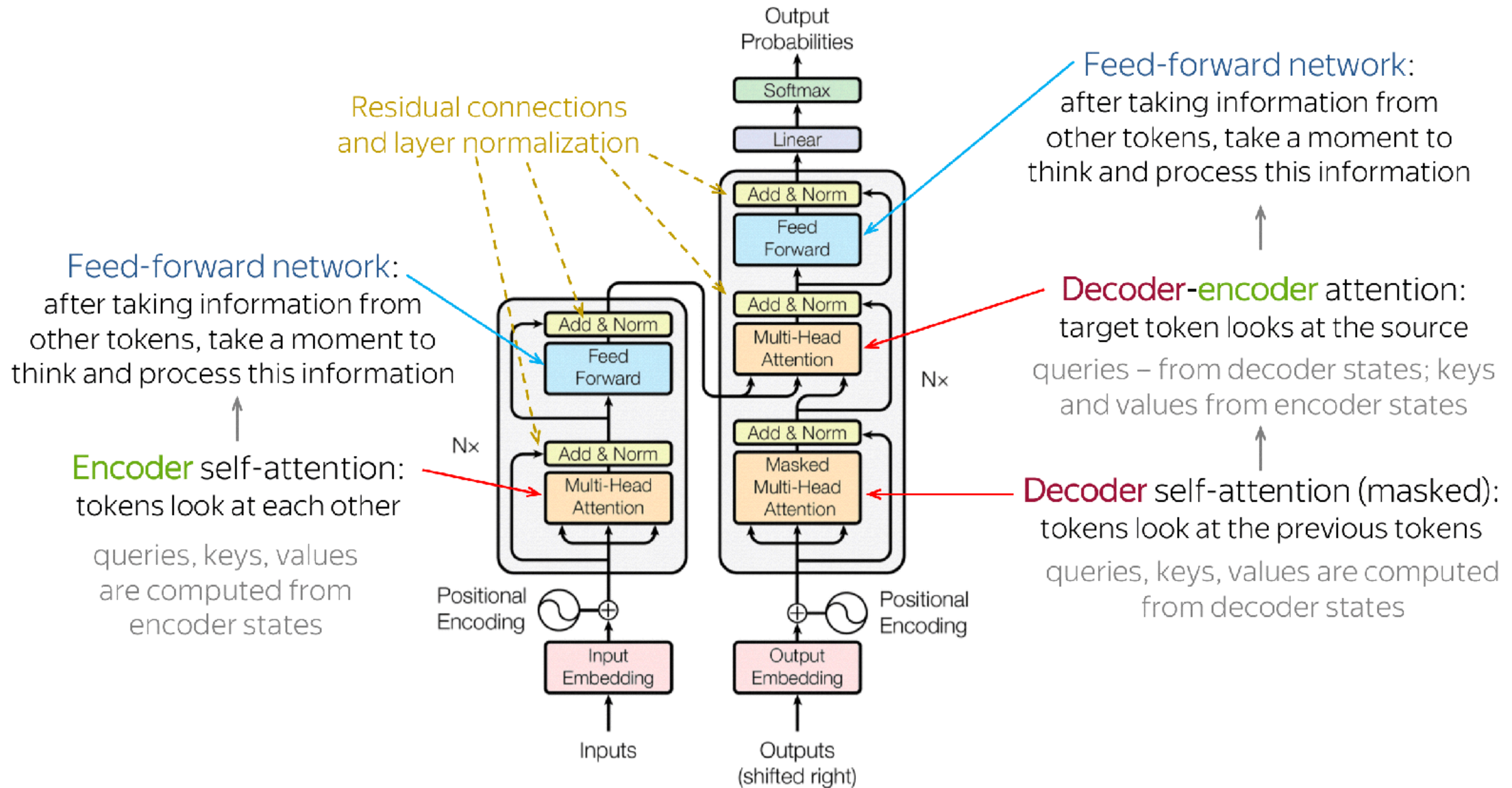
# Multi-Head Attention

## Independently Focus on Different Things

- Formally, this is implemented as several attention mechanisms whose results are combined:

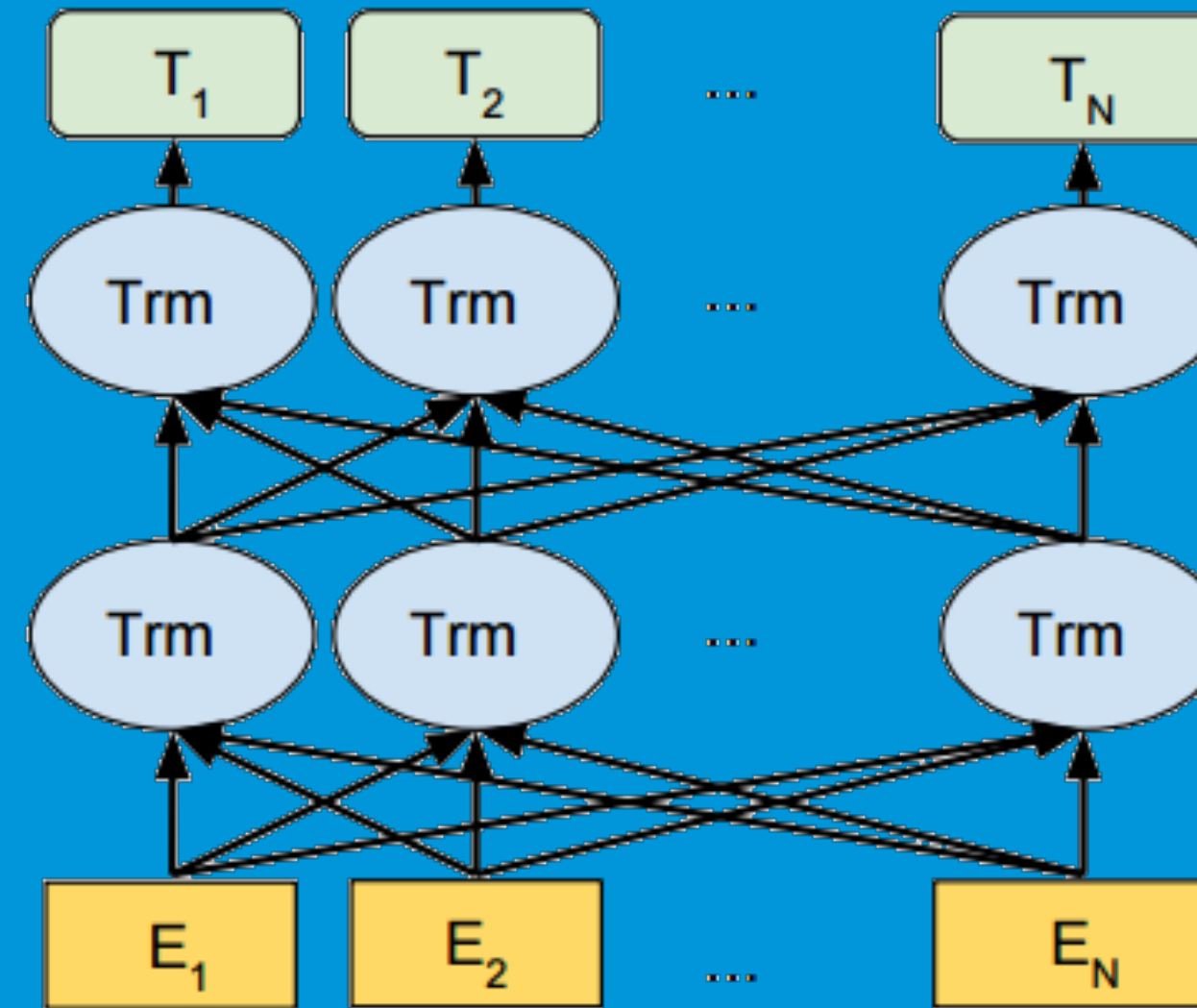


# Transformer: Model Architecture





# BERT



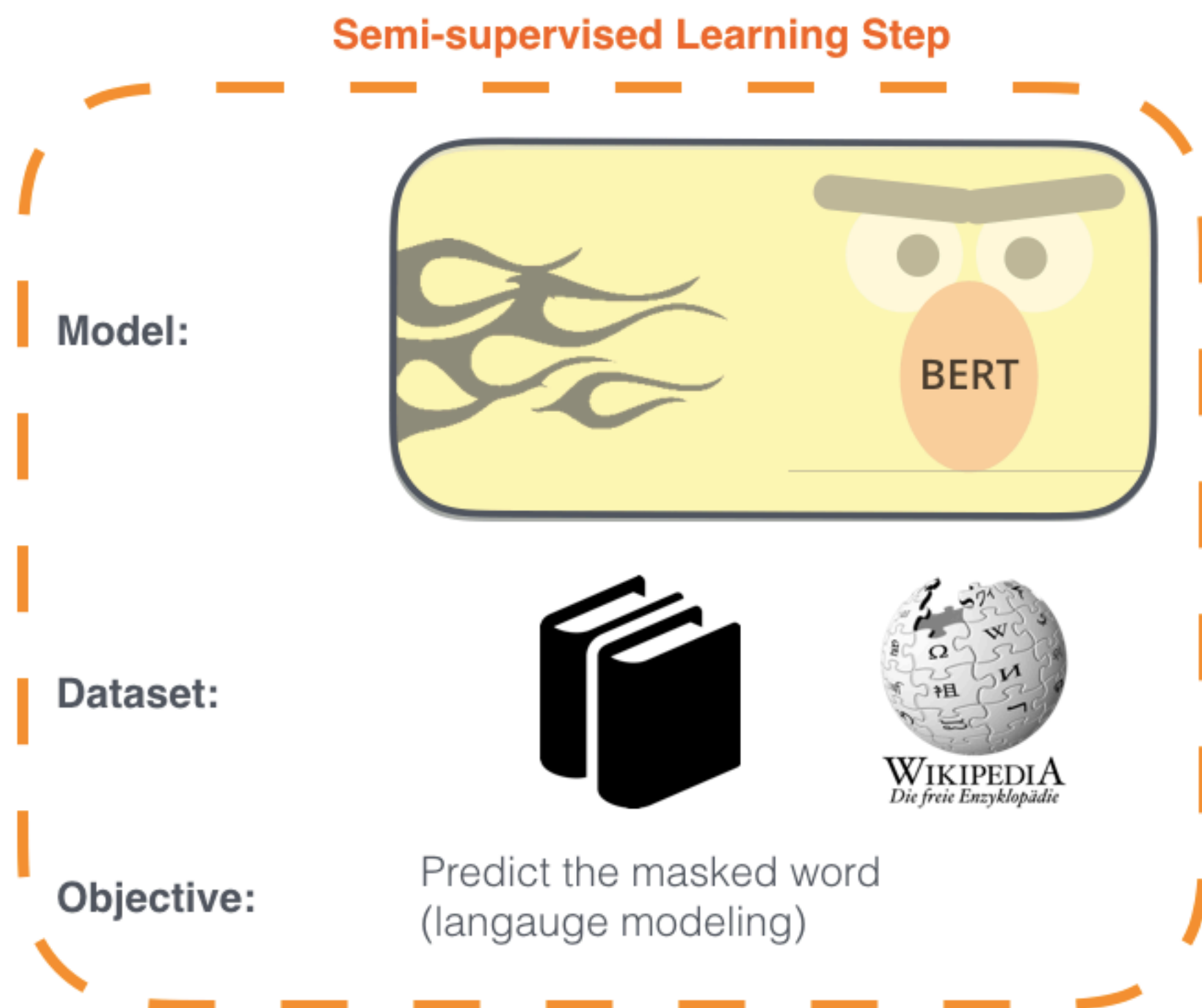
**BERT:**  
**B**idirectional **E**ncoder  
**R**epresentations from  
**T**ransformers



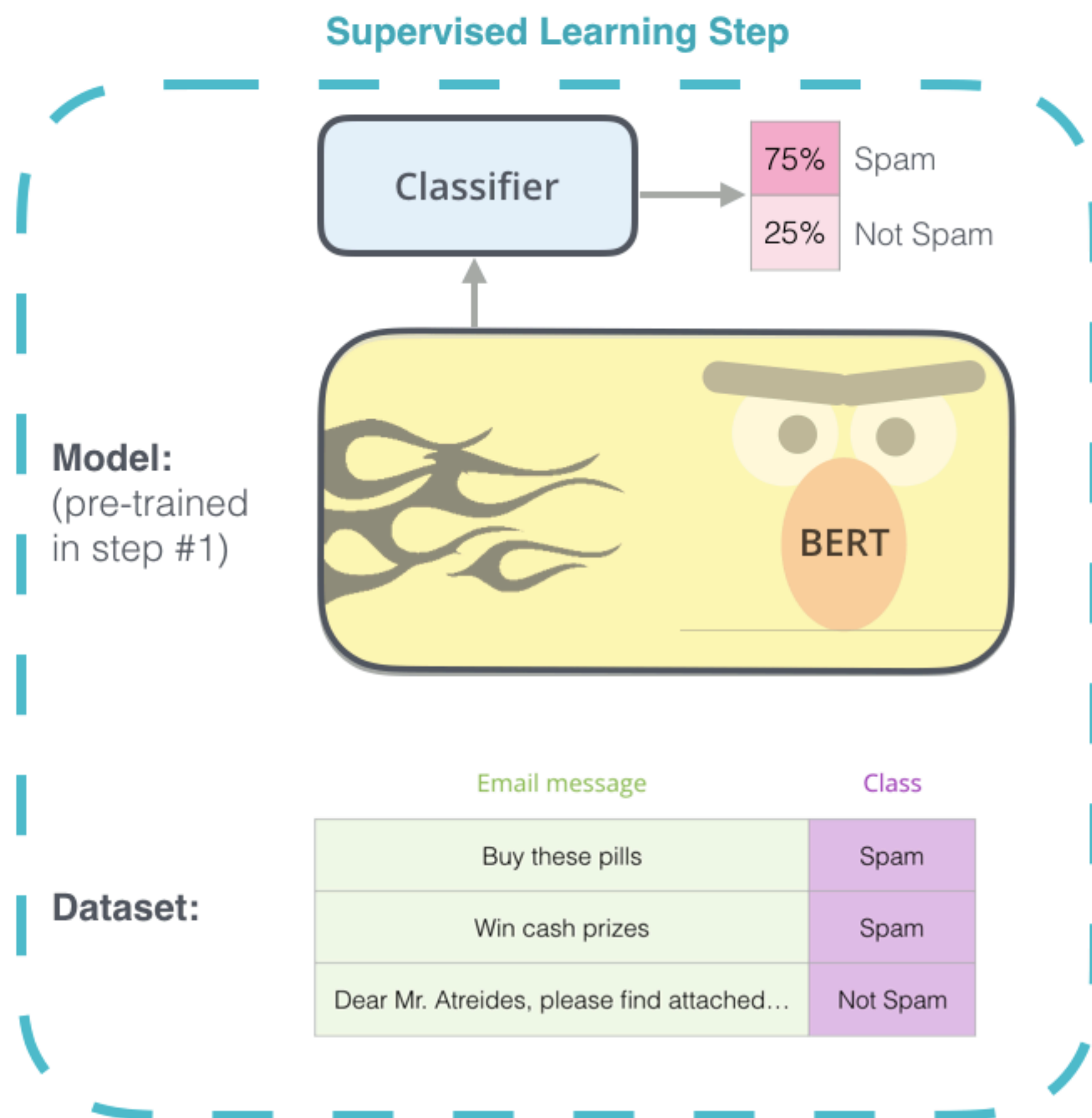
# BERT Overview

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [Source for book icon].

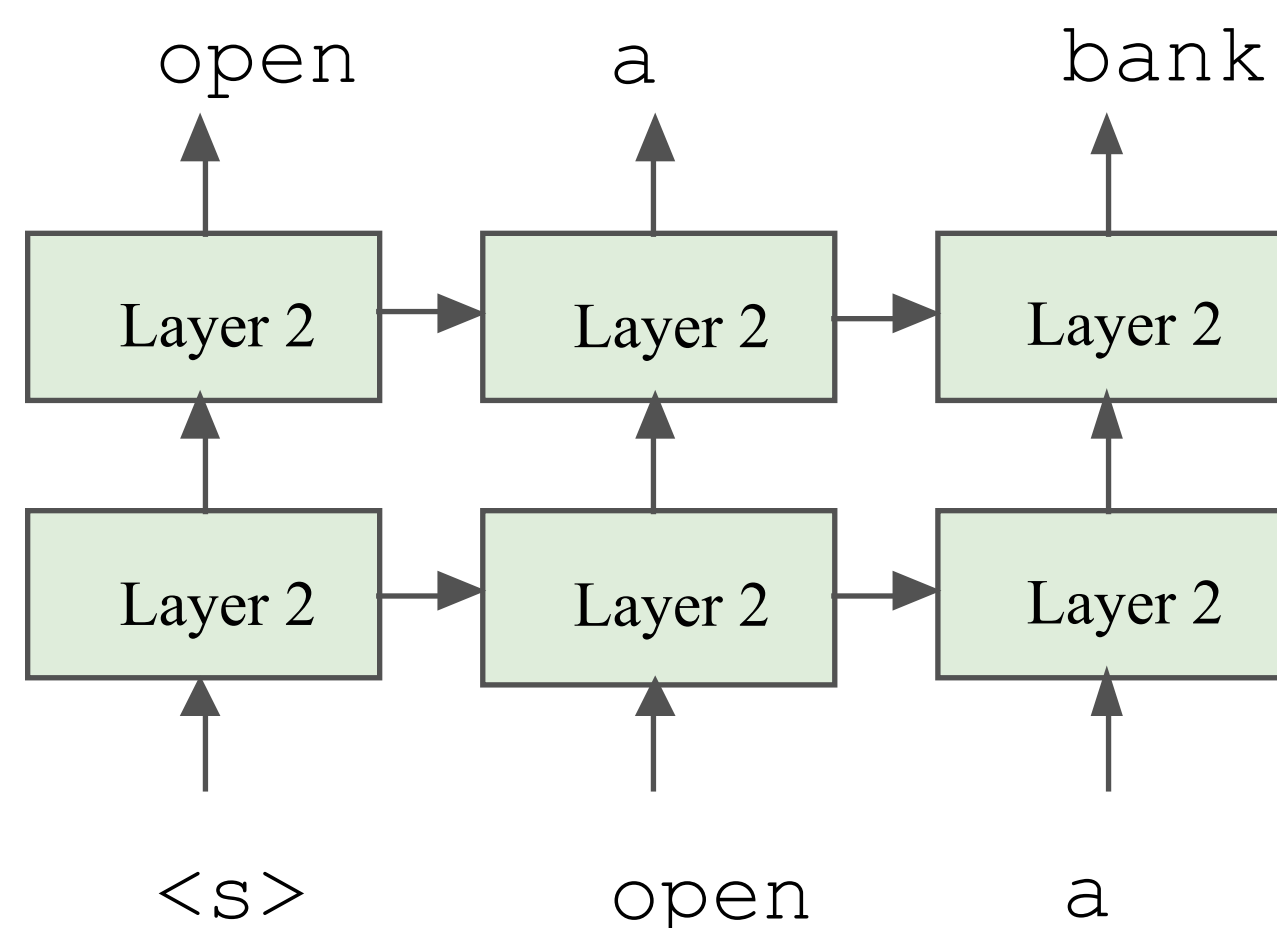
# Problem with Previous Methods

- ⦿ **Problem:** Language models only use left context or right context, but language understanding is bidirectional.
- ⦿ Why are LMs unidirectional?
  - Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - Reason 2: Words can “see themselves” in a bidirectional encoder.

# 97 Unidirectional vs. Bidirectional Models

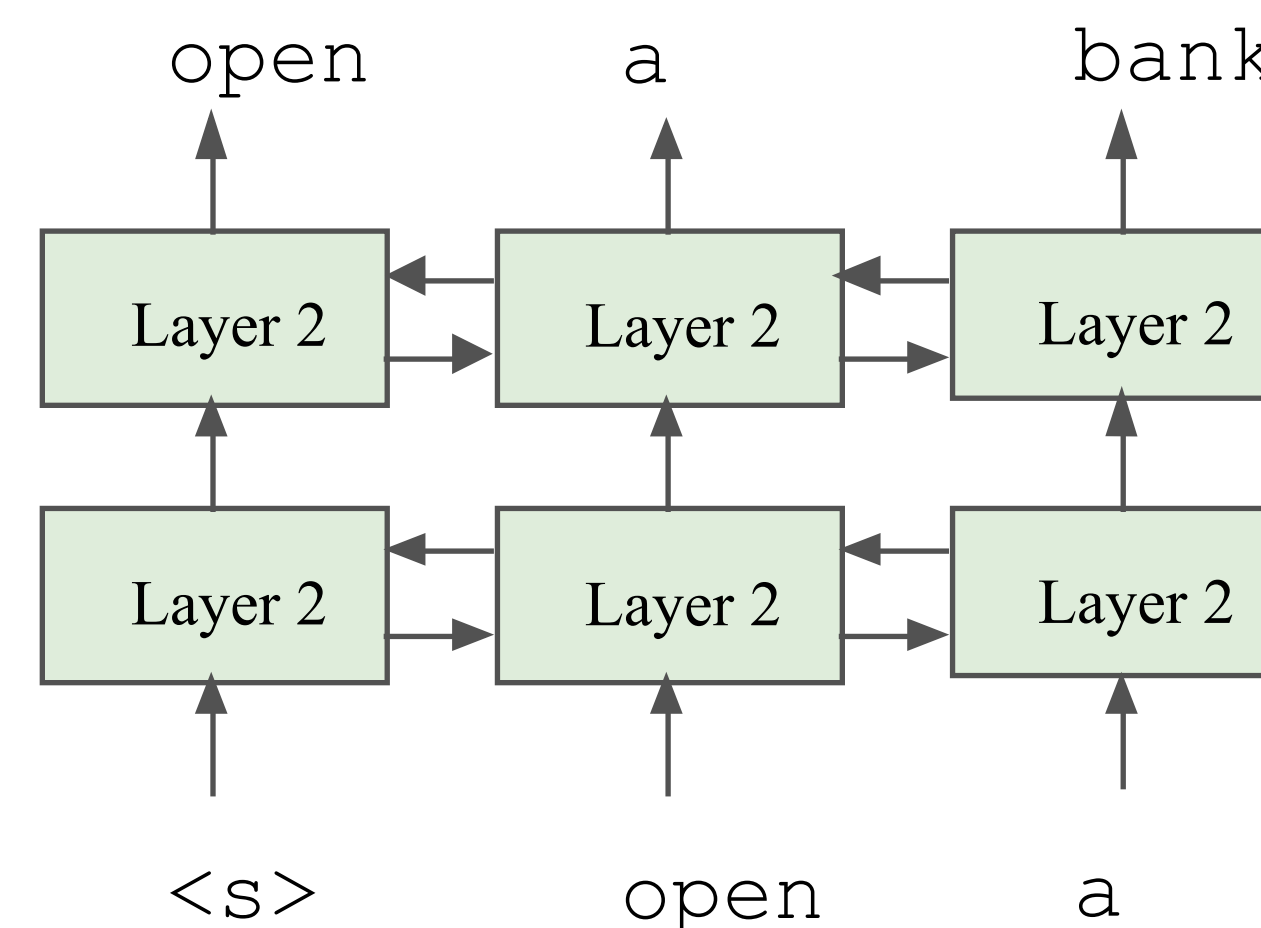
## Unidirectional context

Build representation incrementally



## Bidirectional context

Words can “see themselves”



## 98 Masked LM

- **Solution:** Mask out  $k\%$  of the input words, and then predict the masked words (use 15%)

the man went to the [MASK] to buy a [MASK] of milk

store                      gallon

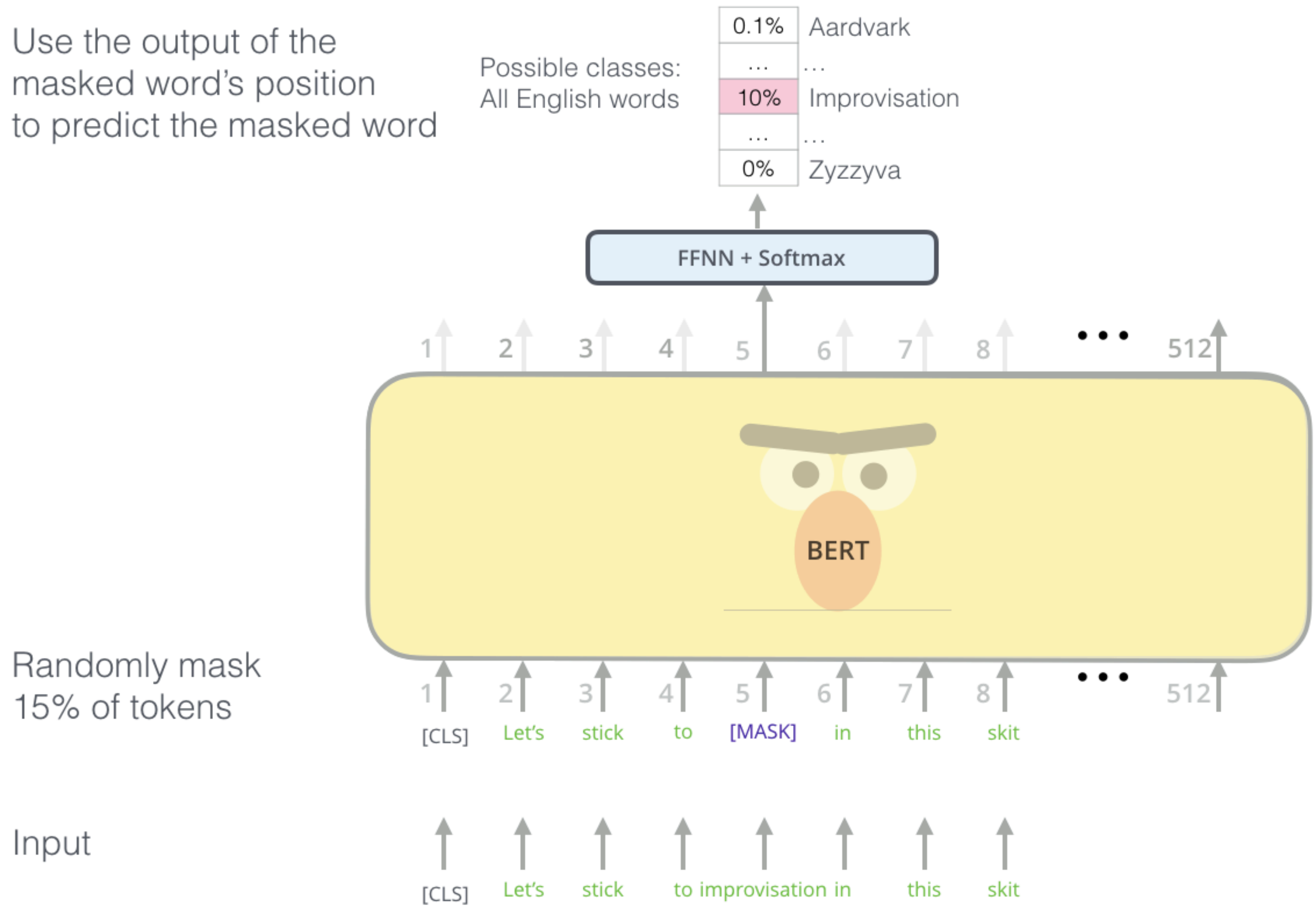
↑                              ↑

- Too little masking: Too expensive to train
- Too much masking: Not enough context



# 99 Masked LM

Use the output of the masked word's position to predict the masked word



BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

# 100 Masked LM

- **Problem:** Mask token never seen at fine-tuning
- **Solution:** 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, replace with [MASK]  
went to the store → went to the [MASK]
  - 10% of the time, replace random word  
went to the store → went to the running
  - 10% of the time, keep same  
went to the store → went to the store

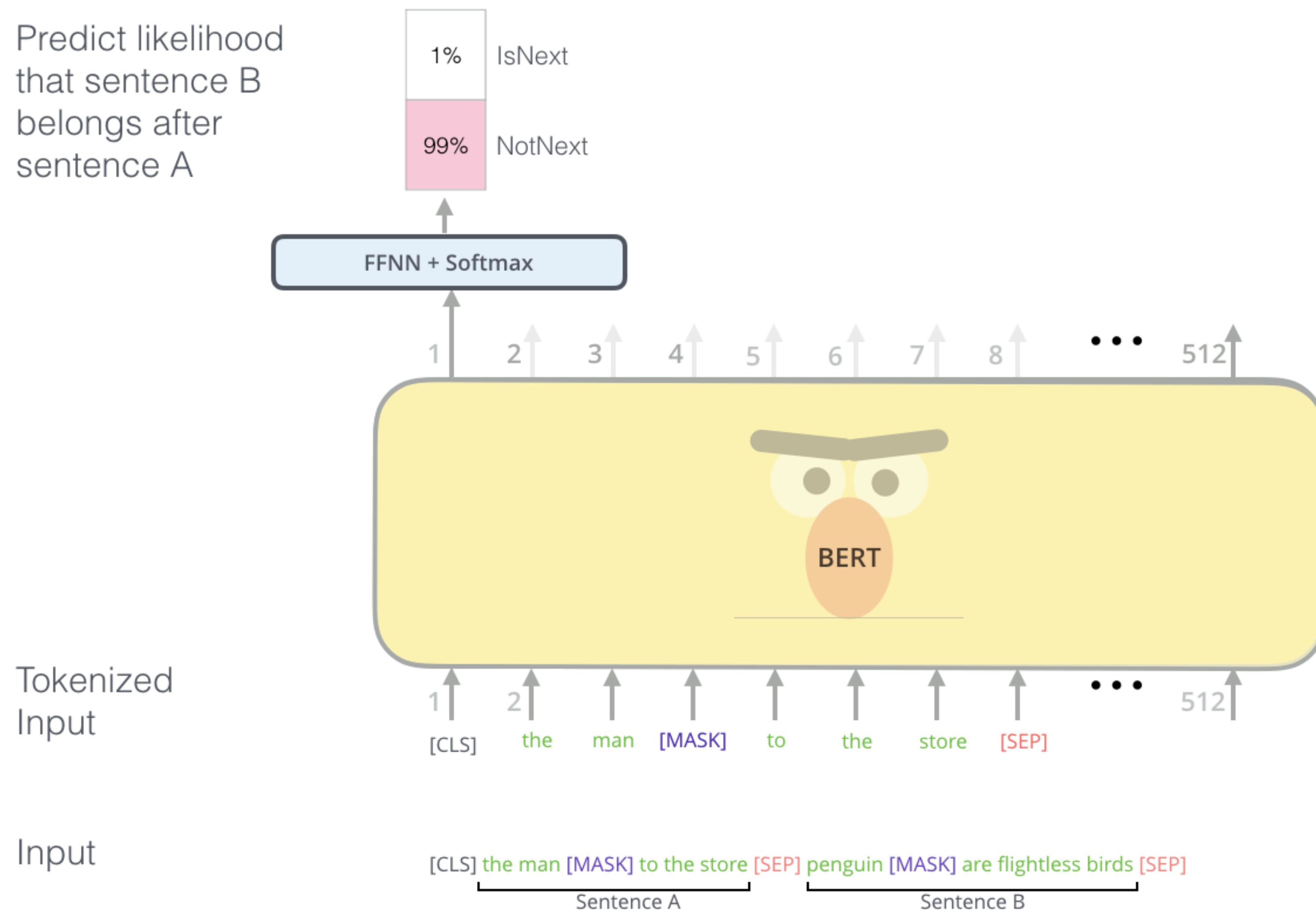
## 101 Next Sentence Prediction

- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

**Sentence A** = The man went to the store.  
**Sentence B** = He bought a gallon of milk.  
**Label** = IsNextSentence

**Sentence A** = The man went to the store.  
**Sentence B** = Penguins are flightless.  
**Label** = NotNextSentence

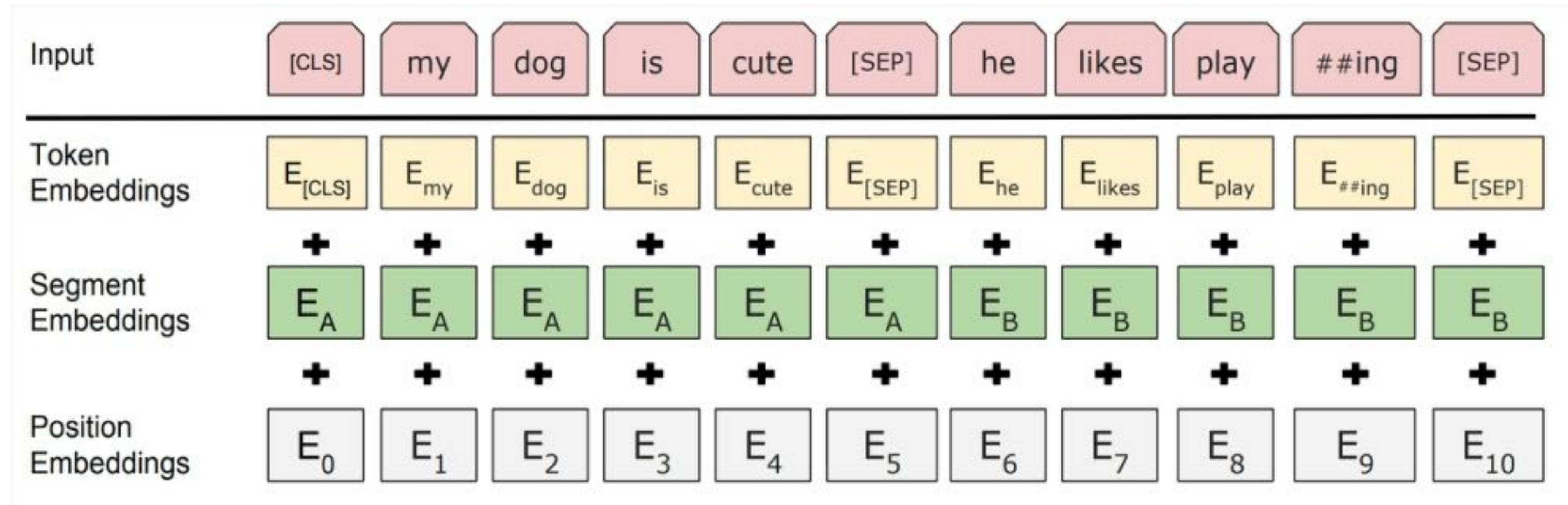
# 102 Next Sentence Prediction



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.



# Input Representation



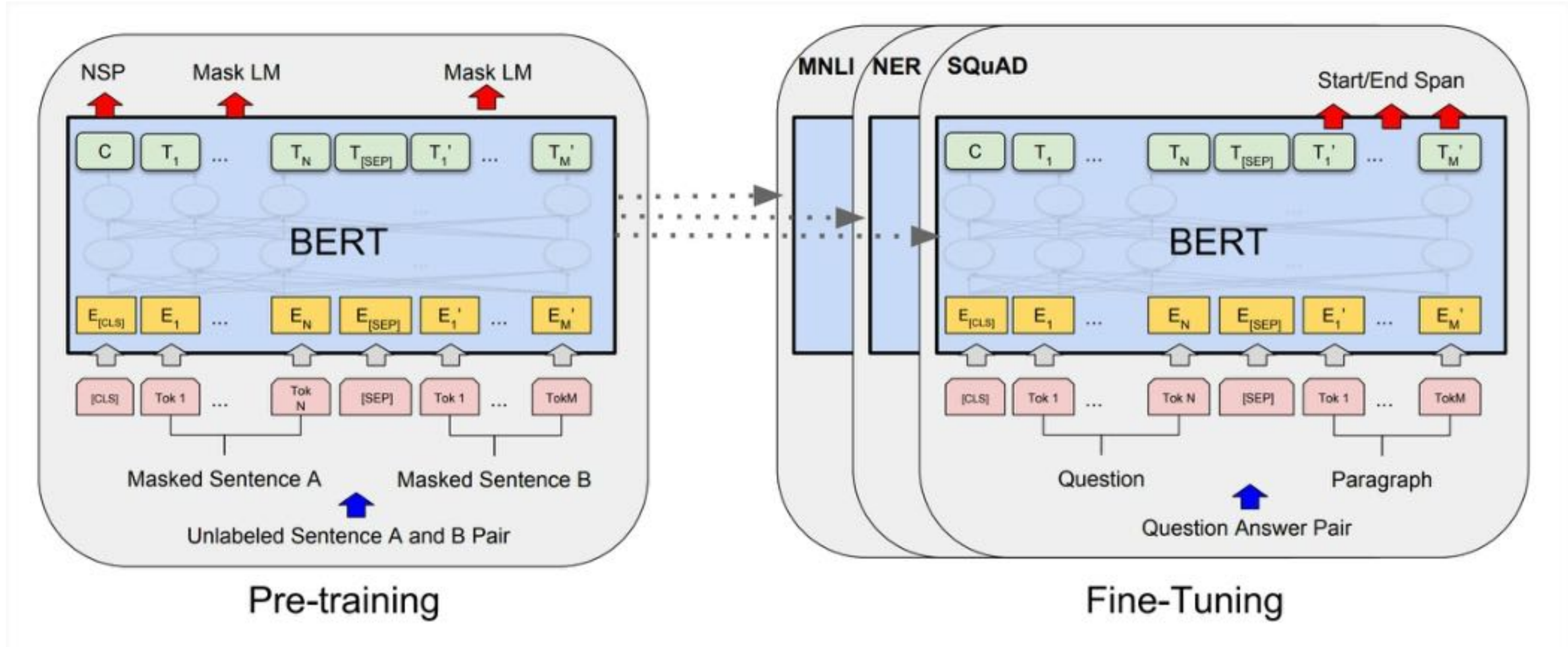
- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings.

## 104 More Details

- **Data:** Wikipedia (2.5B words) + BookCorpus (800M words)
- **Batch Size:** 131,072 words (1024 sequences \* 128 length or 256 sequences \* 512 length)
- **Training Time:** 1M steps (~40 epochs)
- **Optimizer:** AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- **Trained on** 4x4 or 8x8 TPU slice for 4 days

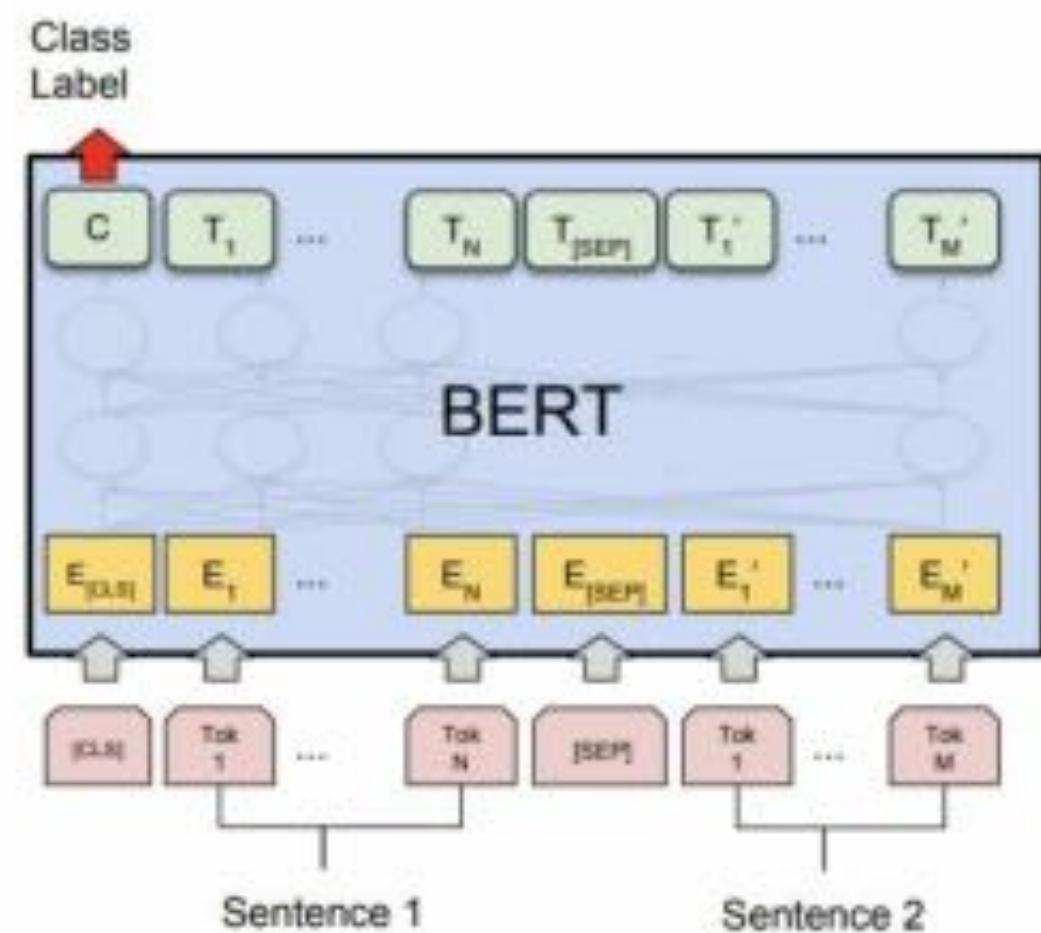


# 105 Fine-Tuning Procedure

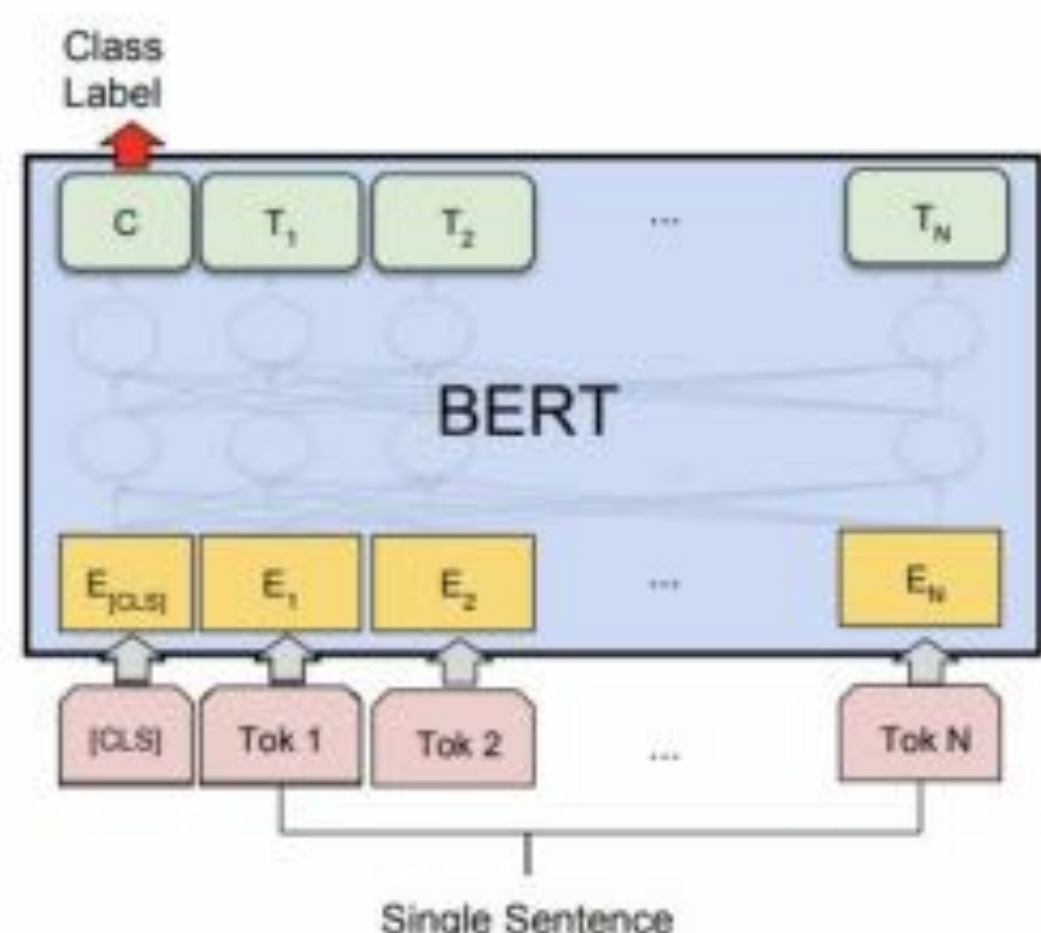




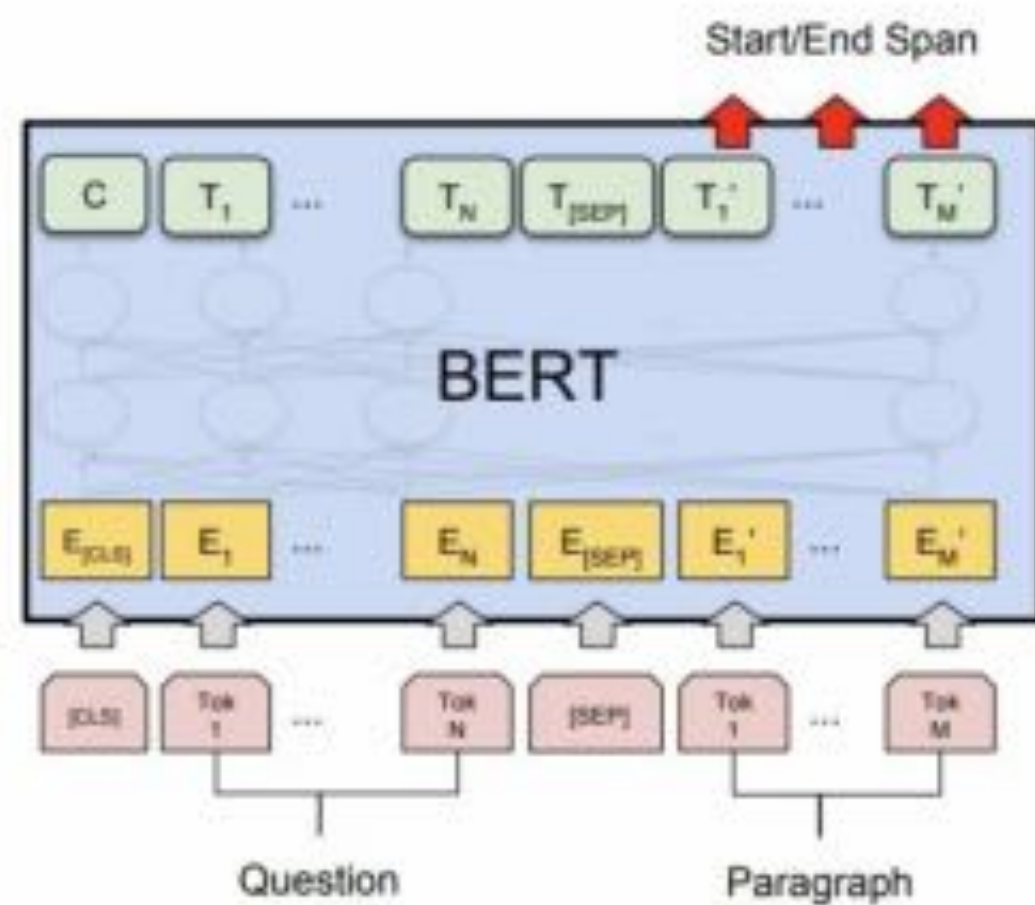
# Fine-Tuning Procedure



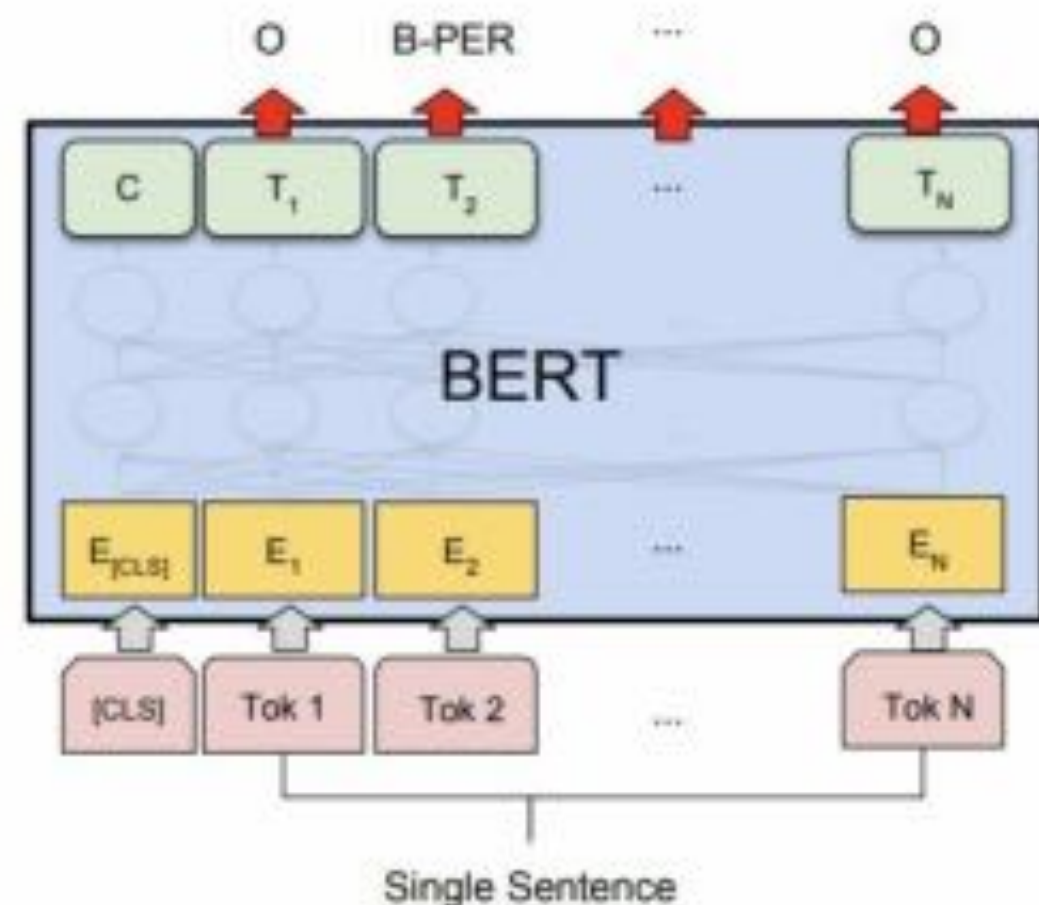
(a) Sentence Pair Classification Tasks: MNLi, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



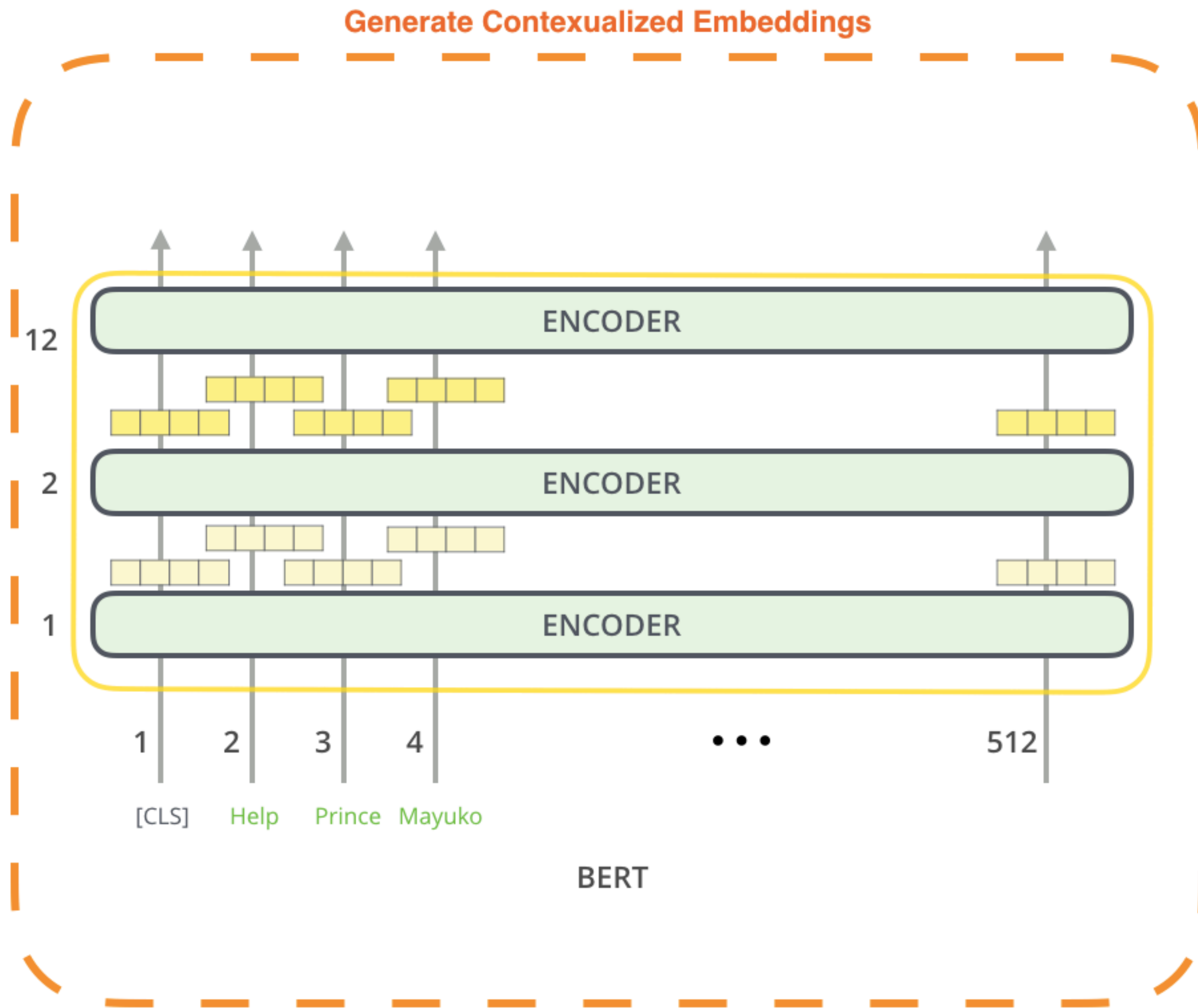
(c) Question Answering Tasks: SQuAD v1.1



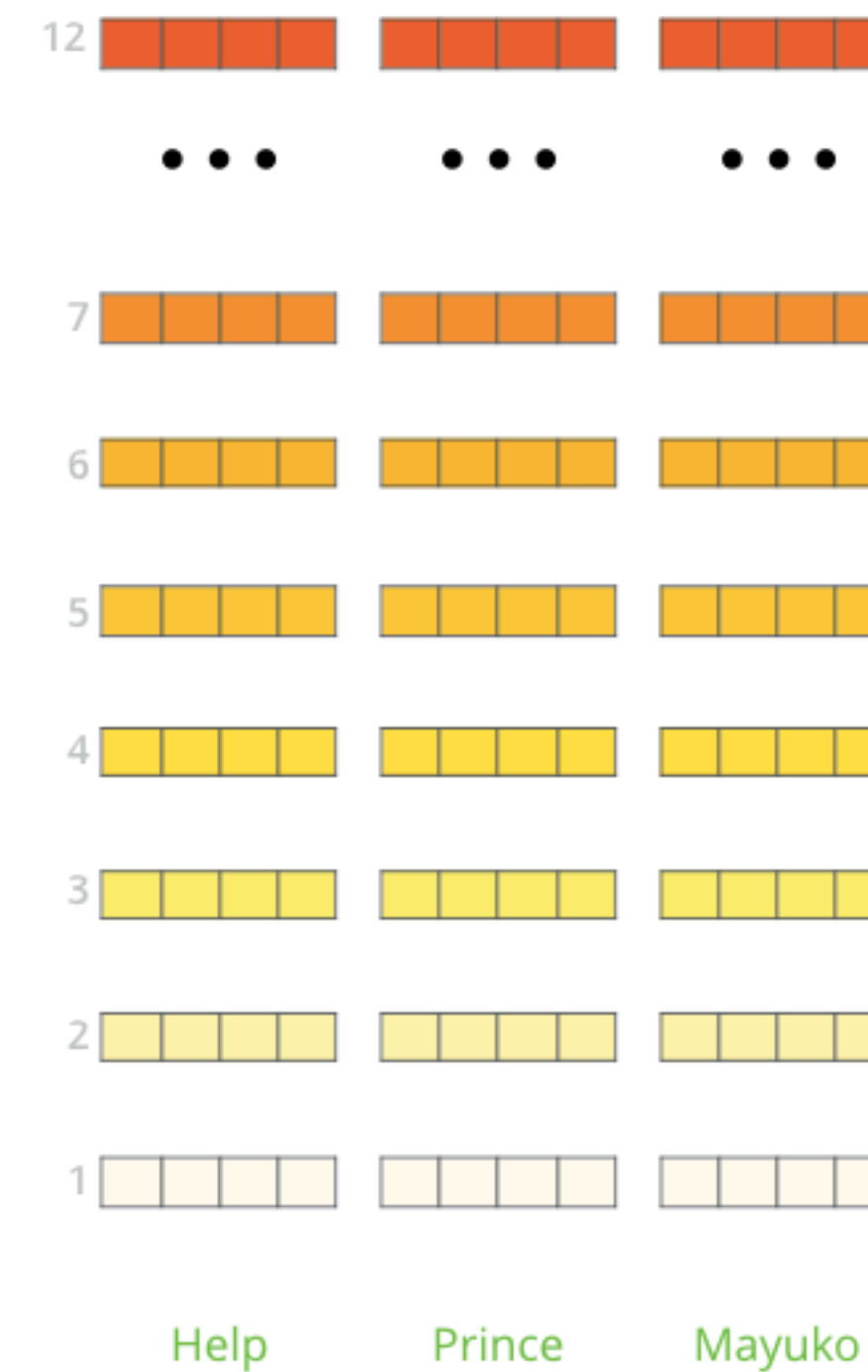
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER



# BERT for Feature Extraction



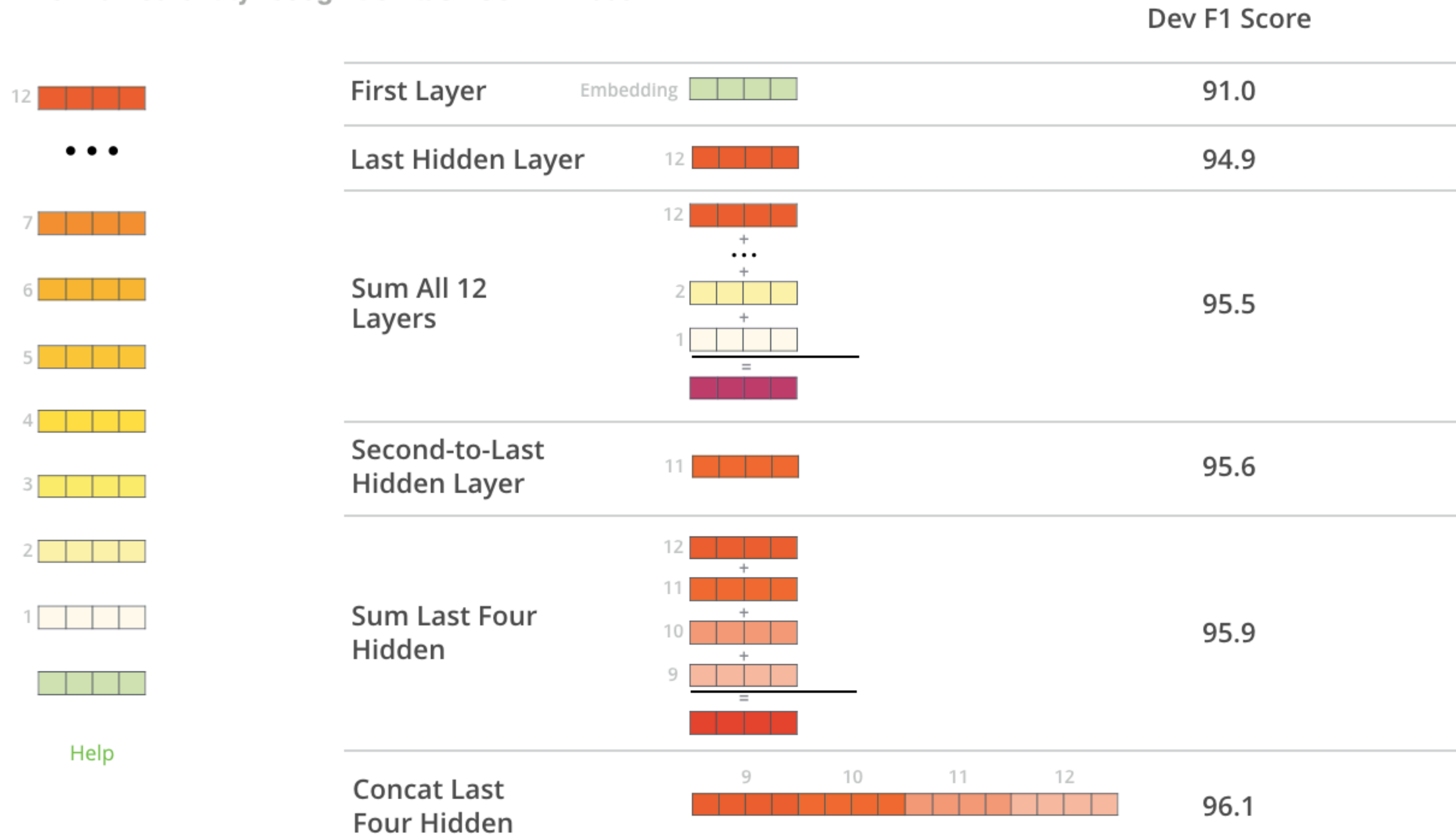
The output of each encoder layer along each token's path can be used as a feature representing that token.



**But which one should we use?**

# BERT for Feature Extraction

What is the best contextualized embedding for “Help” in that context?  
 For named-entity recognition task CoNLL-2003 NER



Help

## Performance: GLUE



ML<sup>2</sup>



DeepMind

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems. GLUE consists of:

- A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty,
- A diagnostic dataset designed to evaluate and analyze model performance with respect to a wide range of linguistic phenomena found in natural language, and
- A public leaderboard for tracking performance on the benchmark and a dashboard for visualizing the performance of models on the diagnostic set.

Corpus	Train	Test	Task	Metrics	Domain
Single-Sentence Tasks					
CoLA	8.5k	<b>1k</b>	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
Similarity and Paraphrase Tasks					
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	<b>391k</b>	paraphrase	acc./F1	social QA questions
Inference Tasks					
MNLI	393k	<b>20k</b>	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	<b>146</b>	coreference/NLI	acc.	fiction books

Table 1: Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form.



## 111 Performance: GLUE

### GLUE Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

# Effect of Model Size



- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have not asymptoted

# **A Few Post-BERT Pre-training Advancements**

# RoBERTa

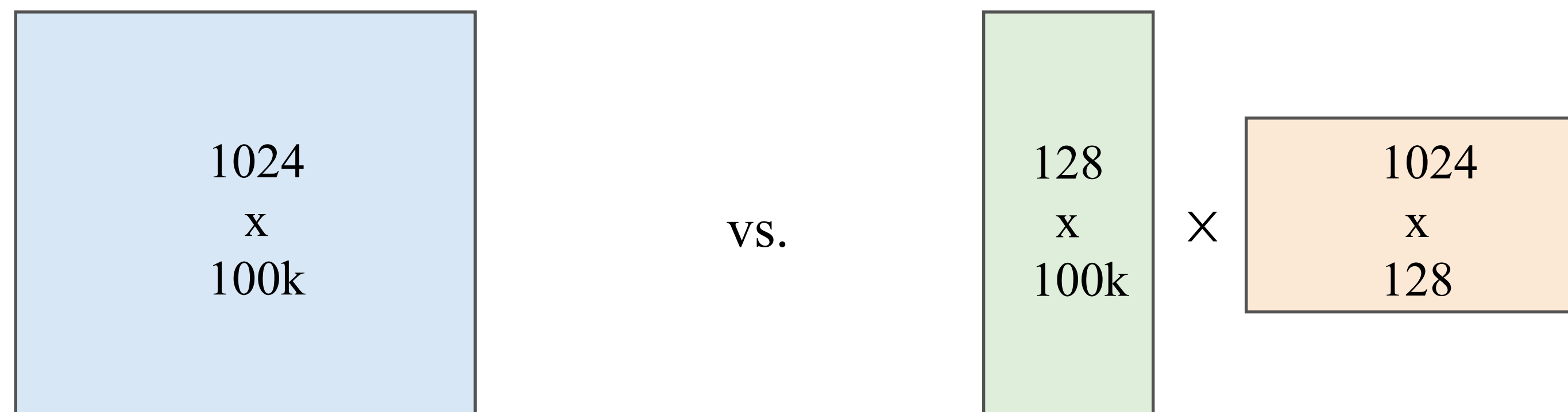
- RoBERTa: A Robustly Optimized BERT Pretraining Approach (Liu et al, University of Washington and Facebook, 2019)
- Trained BERT for more epochs and/or on more data
  - Showed that **more epochs alone helps**, even on same data
  - More data also helps**
- Improved masking and pre-training data slightly

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT <sub>LARGE</sub>	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet <sub>LARGE</sub>	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	<b>90.2/90.2</b>	<b>94.7</b>	<b>92.2</b>	<b>86.6</b>	<b>96.4</b>	<b>90.9</b>	<b>68.0</b>	<b>92.4</b>	<b>91.3</b>	-



# 115 ALBERT

- ALBERT: A Lite BERT for Self-supervised Learning of Language Representations (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: **Factorized embedding parameterization**
  - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix
- Innovation #2: **Cross-layer parameter sharing**
  - Share all parameters between Transformer layers



- Results:

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
<i>Single-task single models on dev</i>								
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	<b>92.2</b>	86.6	96.4	<b>90.9</b>	68.0	92.4
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7
ALBERT (1.5M)	<b>90.8</b>	<b>95.3</b>	<b>92.2</b>	<b>89.2</b>	<b>96.9</b>	<b>90.9</b>	<b>71.4</b>	<b>93.0</b>

- ALBERT is light in terms of parameters, not speed

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	<b>94.1/88.3</b>	<b>88.1/85.1</b>	<b>88.0</b>	<b>95.2</b>	<b>82.3</b>	<b>88.7</b>	0.3x

Jacob Devlin,





- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (Raffel et al, Google, 2019)
- **Ablated many aspects of pre-training:**
  - Model size
  - Amount of training data
  - Domain/cleanness of training data
  - Pre-training objective details (e.g., span length of masked text)
  - Ensembling
  - Finetuning recipe (e.g., only allowing certain layers to finetune)
  - Multi-task training



- Conclusions:

- **Scaling up model size and amount of training data** helps a lot
- Best model is 11B parameters (BERT-Large is 330M), trained on 120B words of cleaned common crawl text
- Exact masking/corruptions strategy doesn't matter that much
- Mostly negative results for better finetuning and multi-task strategies

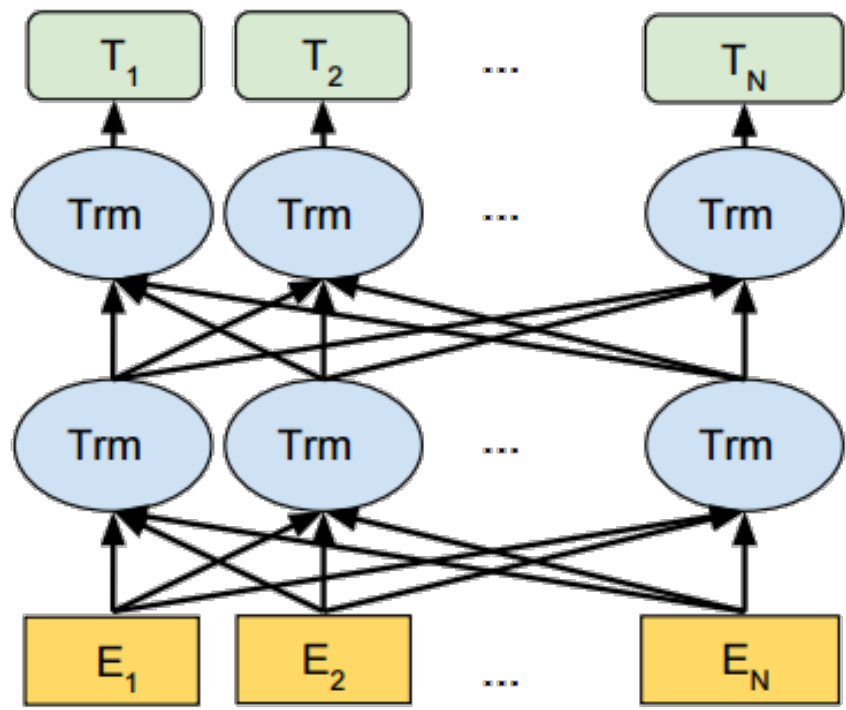
- T5 results on SuperGLUE: <https://super.gluebenchmark.com/leaderboard/>

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultIRC	ReCoRD	RTE	WIC	WSC	AX-b	AX-g	
+	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
+	2	Zirui Wang	T5 + Meena, Single Model (Meena Team - Google Brain)		90.2	91.3	95.8/97.6	97.4	88.3/63.0	94.2/93.5	92.7	77.9	95.9	66.5	88.8/89.9
	3	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	4	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
+	5	Huawei Noah's Ark Lab	NEZHA-Plus		86.7	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	58.0	87.1/74.4

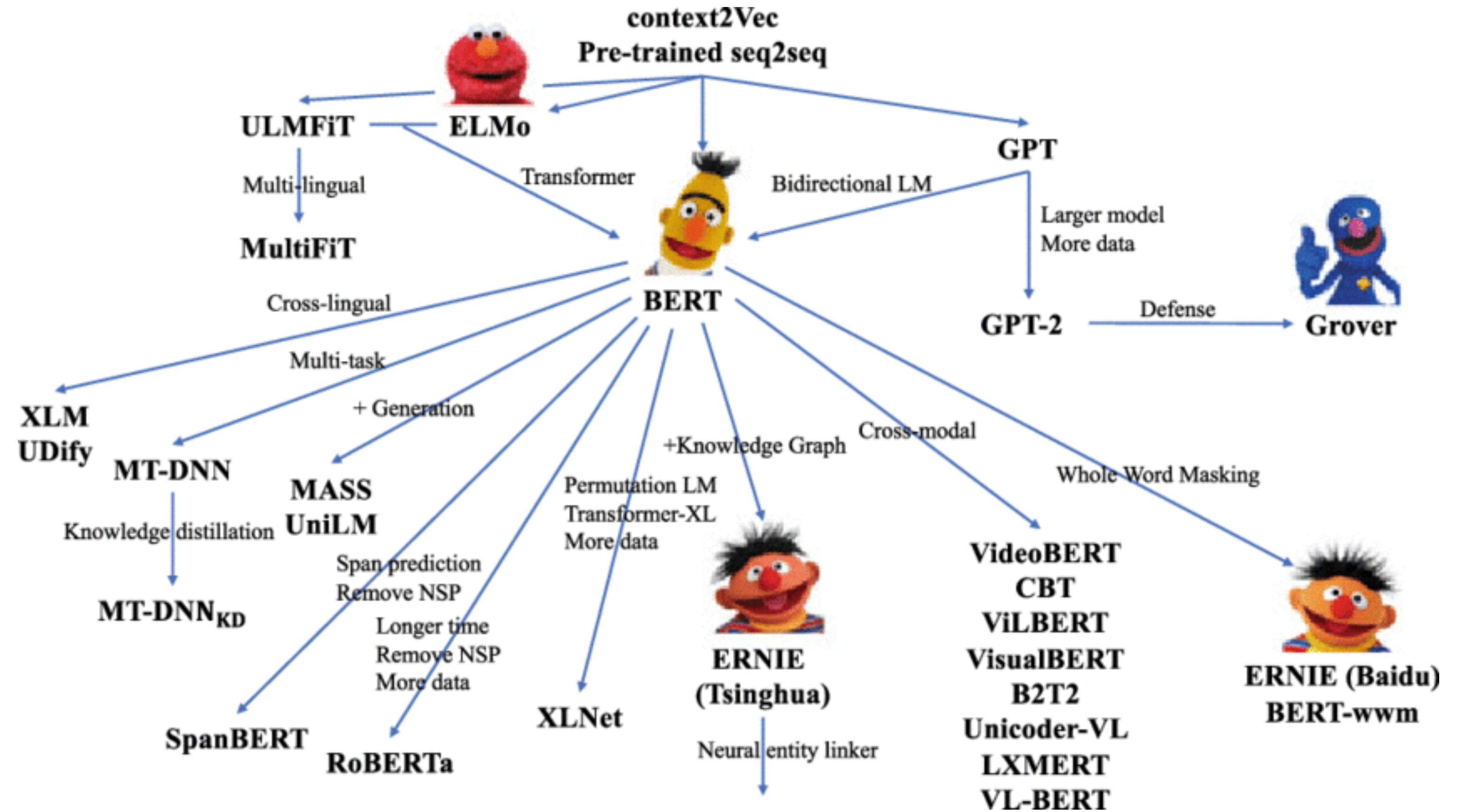
Jacob Devlin,



# 119 Pre-trained language models

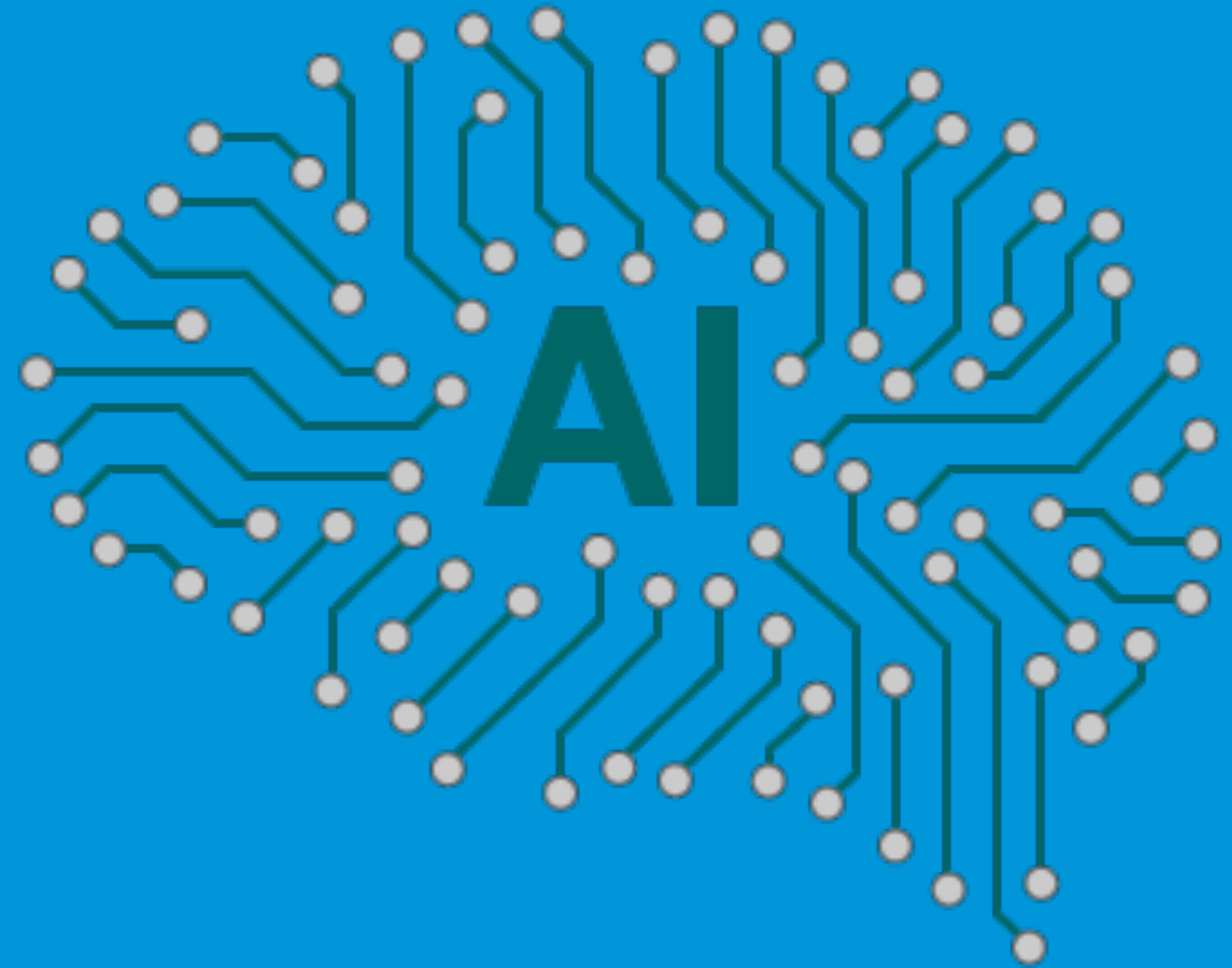


**BERT**



**The Pre-trained language model family**

# AI Research Frontiers





# 121 AlphaCode



Research

## Competitive programming with AlphaCode

February 2, 2022

AlphaCode uses transformer-based language models to generate code at an unprecedented scale, and then smartly filters to a small set of promising programs.

AlphaCode achieved an estimated rank within the top 54% of participants in programming competitions.



# DALL·E 2

DALL·E 2 can create original, realistic images and art from a text description. It can combine concepts, attributes, and styles.

TEXT DESCRIPTION

An astronaut riding a horse in space in a photorealistic style

Teddy bears playing basketball with cats in space

A bowl of soup in the style of Andy Warhol as a pencil drawing



DALL·E 2



DALL·E 2 is a new AI system that can create realistic images and art from a description in natural language.





an espresso machine that makes coffee from human souls, artstation



panda mad scientist mixing sparkling chemicals, artstation



a corgi's head depicted as an explosion of a nebula



a dolphin in an astronaut suit on saturn, artstation



a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese



a teddybear on a skateboard in times square

**A two-stage model: a prior that generates a CLIP image embedding given a text caption, and a decoder that generates an image conditioned on the image embedding.**



## 124 Pathways Language Model (PaLM)

### Explaining a joke

#### Prompt

Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

#### Model Response

Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

**Pathways: a single model that could generalize across domains and tasks while being highly efficient.**

**PaLM: a 540-billion parameter, dense decoder-only Transformer model trained with the Pathways system, which enabled us to efficiently train a single model across multiple TPU v4 Pods. SOTA performance across on hundreds of language understanding and generation tasks.**

PaLM explains an original joke with two-shot prompts.

# 125 Pathways Language Model (PaLM)

Prompt

```
// Translate from C to Python
int add_one ( int x ){
  int m = 1;
  while ( x & m ) {
    x = x ^ m;
    m <<= 1;
  }
  x = x ^ m;
  return x; }
```

Model Response

Examples of a fine-tuned PaLM 540B model on text-to-code tasks



# 126 Use AI to control plasmas for nuclear fusion



Research

## Accelerating fusion science through learned plasma control

February 16, 2022

Develop the first deep reinforcement learning (RL) system to autonomously discover how to control these coils and successfully contain the plasma in a tokamak, opening new avenues to advance nuclear fusion research.



# Thanks! Q&A