

Transformers architecture

Large Language Models

Intro to modern natural language processing

Part II

Maxime LOUIS

Research Scientist

Naver LABS Europe

maxime.louis@naverlabs.com

Course adapted from:

- Nadia Chirkova and Laurent Besacier NLP Course at Centrale Paris

(Huge thanks to Nadia)

- Stanford transformers course

Other great resources:

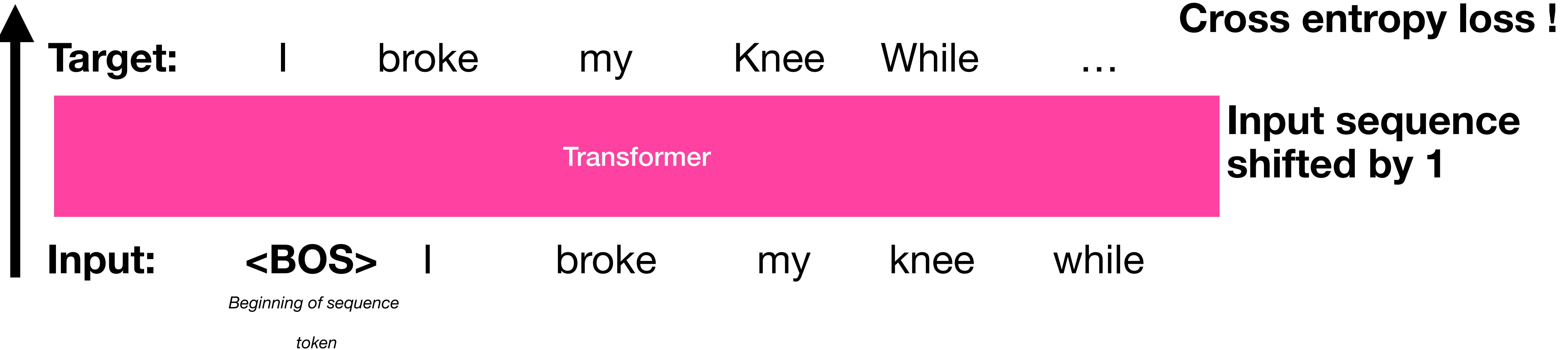
<https://www.youtube.com/watch?v=wjZofJX0v4M&t> (2Blue1Brown)

<https://www.youtube.com/watch?v=kCc8FmEb1nY&t> (openAI employee)

Recap

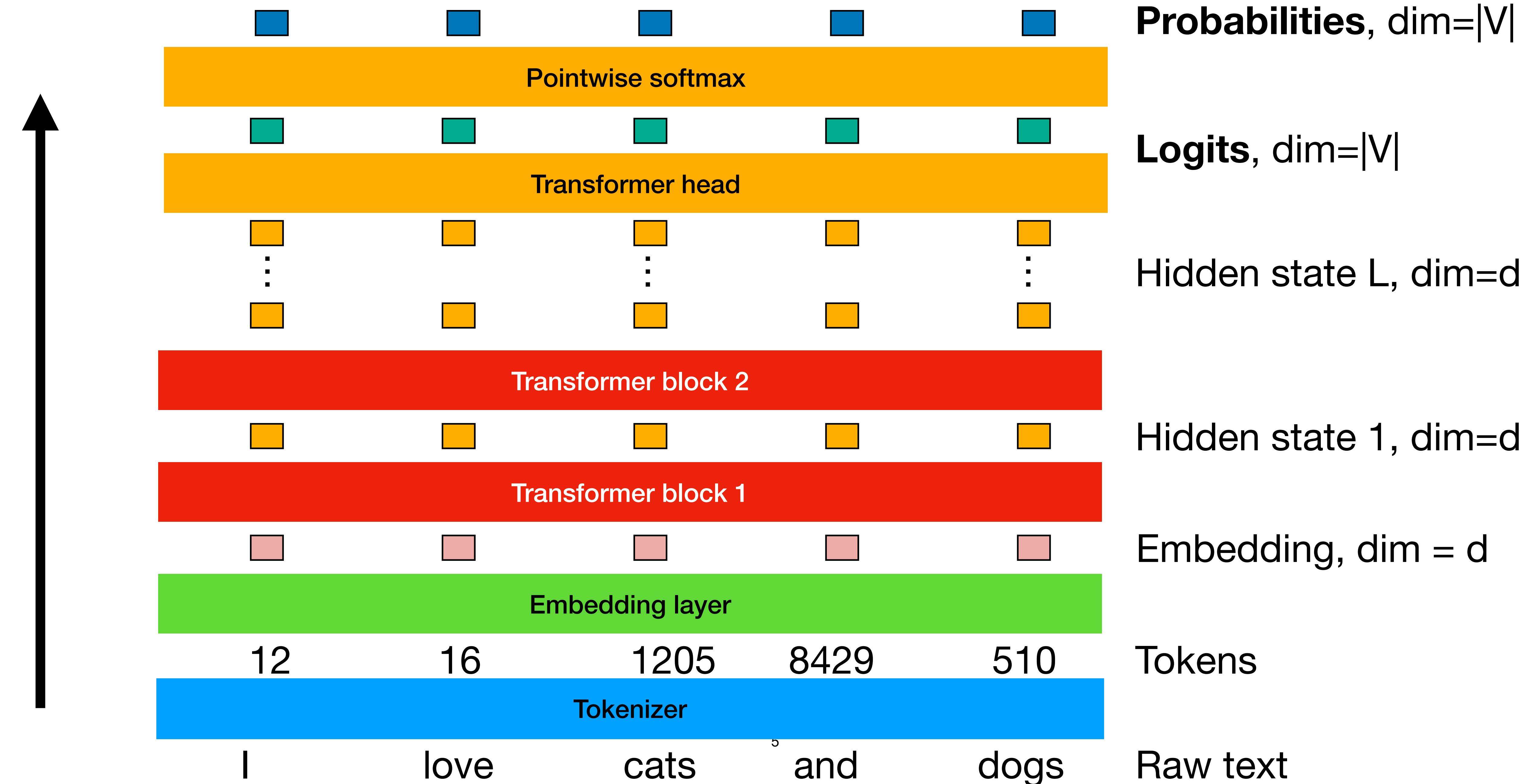
Training objective for transformer

Causal Language Modeling

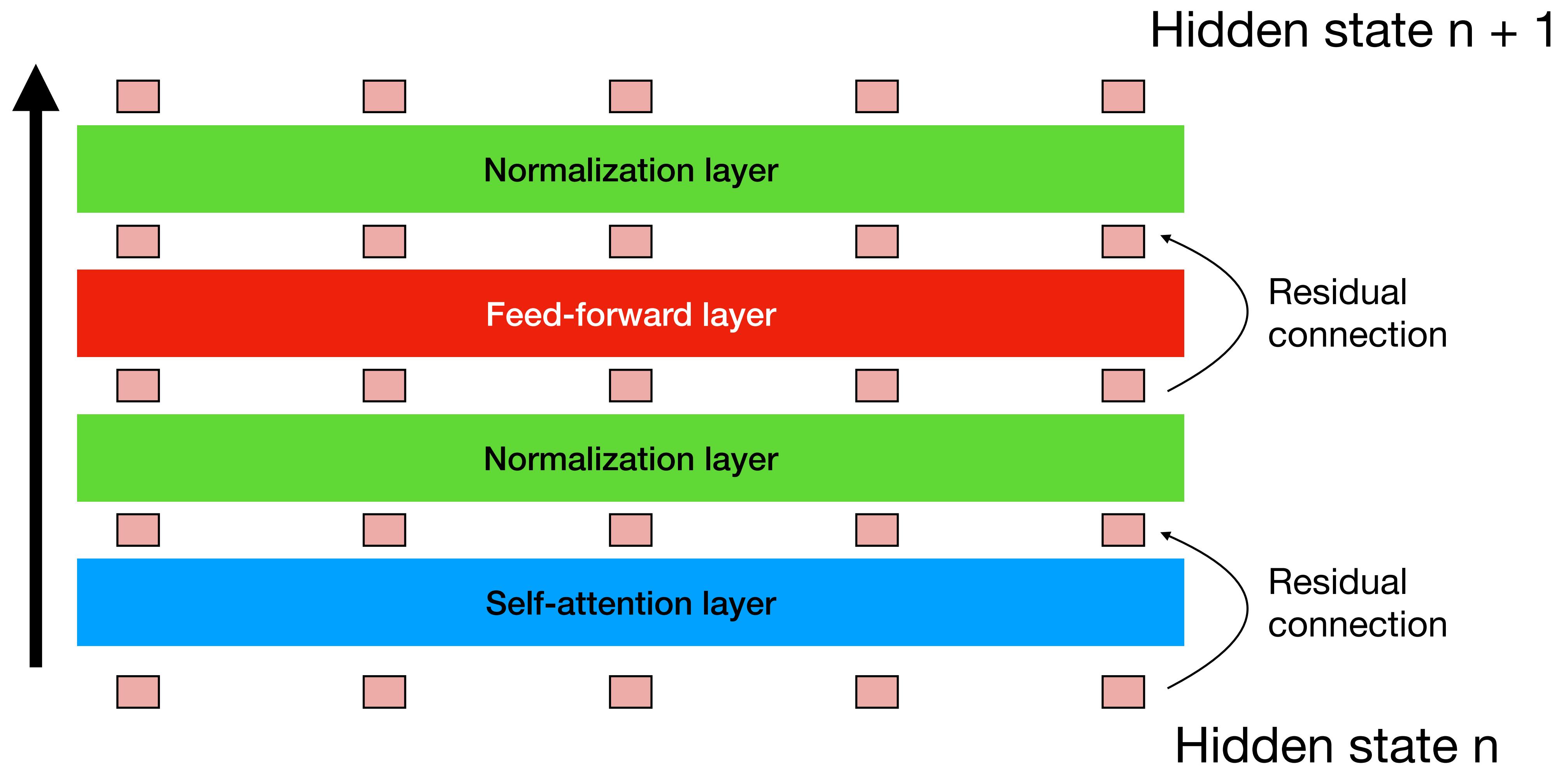


Next-token predictions are made at every position simultaneously

Transformer



Inside a Transformer block



Attention layer

- Step 2:

- Compute all pairs of scalar products between keys and queries
- **$K_i \cdot Q_i$ measure how much a token attends to another token**

	Q_1	Q_2	Q_3
K_1	$K_1 \cdot Q_1$	$K_1 \cdot Q_2$	$K_1 \cdot Q_3$
K_2	$K_2 \cdot Q_1$	$K_2 \cdot Q_2$	$K_2 \cdot Q_3$
K_3	$K_3 \cdot Q_1$	$K_3 \cdot Q_2$	$K_3 \cdot Q_3$

Attention layer

- Step 2:

- Compute all pairs of scalar products between keys and queries
- **$K_i \cdot Q_i$ measure how much a token attends to another token**

	Q_1	Q_2	Q_3
K_1	$K_1 \cdot Q_1$	$K_1 \cdot Q_2$	$K_1 \cdot Q_3$
K_2	$K_2 \cdot Q_1$	$K_2 \cdot Q_2$	$K_2 \cdot Q_3$
K_3	$K_3 \cdot Q_1$	$K_3 \cdot Q_2$	$K_3 \cdot Q_3$

Attention layer

- **Step 2:**

- Compute all pairs of scalar products between keys and queries
- **$K_i \cdot Q_i$ measure how much a token attends to another token**

	Q_1	Q_2	Q_3
K_1	50	-1	3
K_2	3.22	0.99	6
K_3	-0.39	22	43

Attention layer

- **Step 3:**

- If we do CAUSAL language modeling: we need past tokens to NOT attend future tokens

	Q_1	Q_2	Q_3
K_1	50	MASK	MASK
K_2	3.22	0.99	MASK
K_3	-0.39	22	43

Attention layer

- Step 4:
 - Normalize these scores using softmax (row-wise)
 - **Technicality:** scores are divided by \sqrt{d} before softmax to prevent too large logits.

	Q_1	Q_2	Q_3
K_1	1	MASK	MASK
K_2	0.8	0.2	MASK
K_3	0.01	0.6	0.39

The attention layer: summary

Layer output

$$\sum_{i=1}^l \text{softmax}\left(\frac{k_1^\top q_i}{\sqrt{d}}\right) v_i$$

$$\sum_{i=1}^l \text{softmax}\left(\frac{k_1^\top q_i}{\sqrt{d}}\right) v_i$$

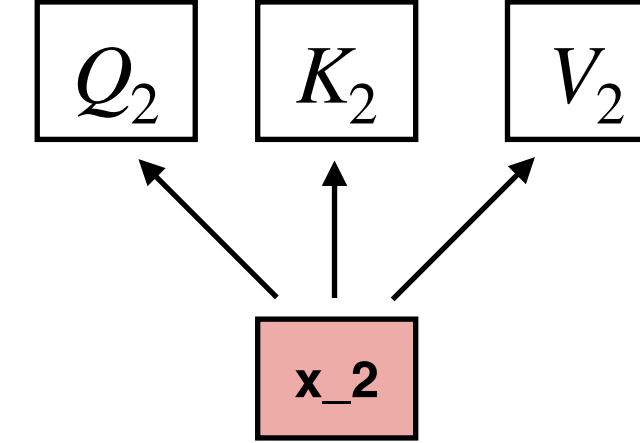
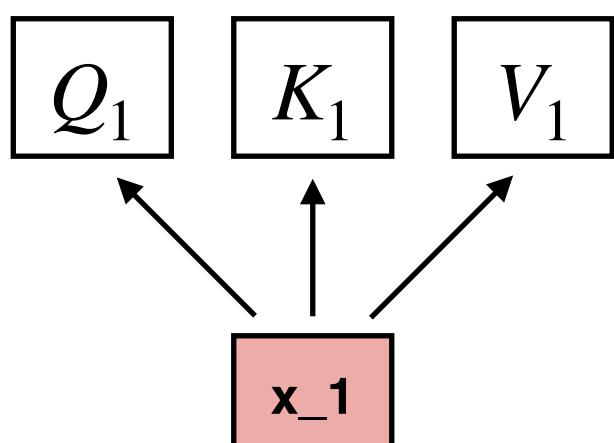
**Normalized
Attention scores**

$$\text{softmax}\left(\frac{k_1^\top q_1}{\sqrt{d}}, \dots, \frac{k_1^\top q_n}{\sqrt{d}}\right) \quad \text{softmax}\left(\frac{k_2^\top q_1}{\sqrt{d}}, \dots, \frac{k_2^\top q_n}{\sqrt{d}}\right)$$

Attention scores

$$\left(\frac{k_1^\top q_1}{\sqrt{d}}, \dots, \frac{k_1^\top q_n}{\sqrt{d}}\right)$$

$$\left(\frac{k_2^\top q_1}{\sqrt{d}}, \dots, \frac{k_2^\top q_n}{\sqrt{d}}\right)$$



$$Q_i = W_Q x_i$$
$$K_i = W_K x_i$$
$$V_i = W_V x_i$$

W_Q, W_K, W_V
are the
parameters
of the layer

**It's also called Scaled
Dot-Product attention**

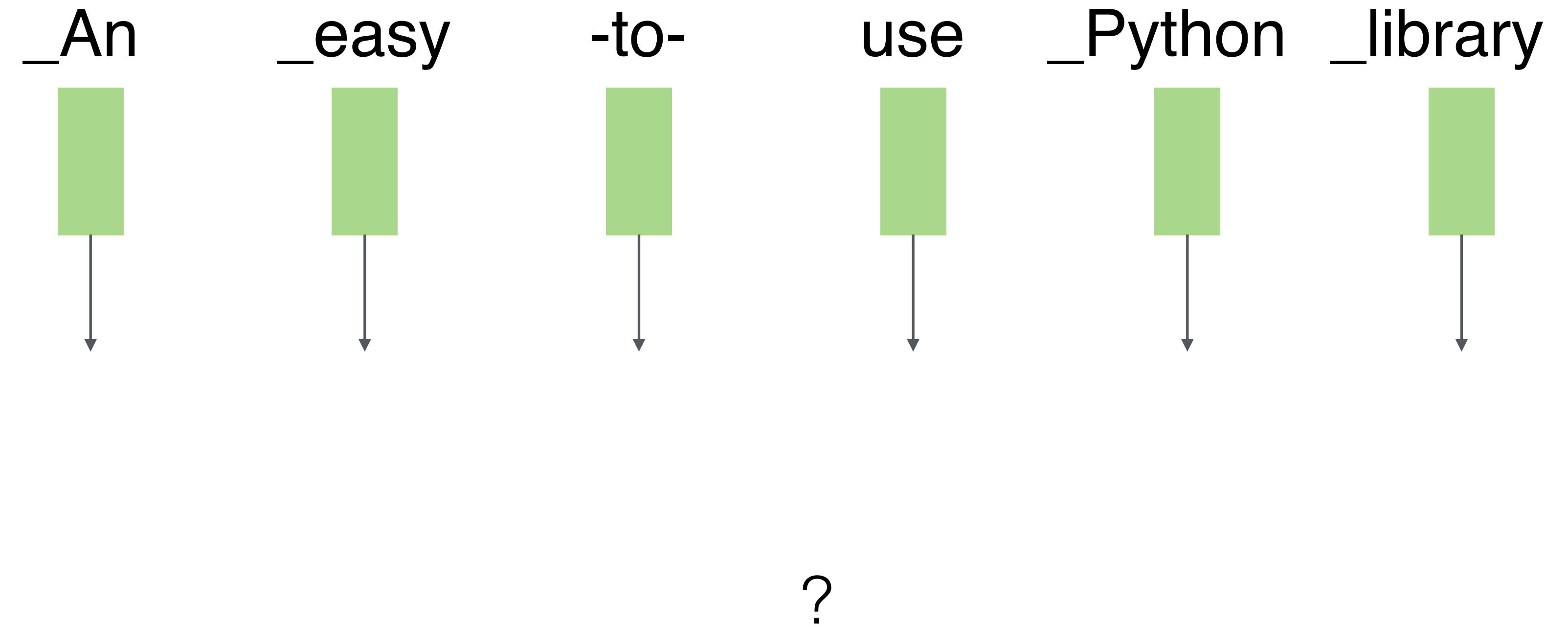
Course outline

Part II

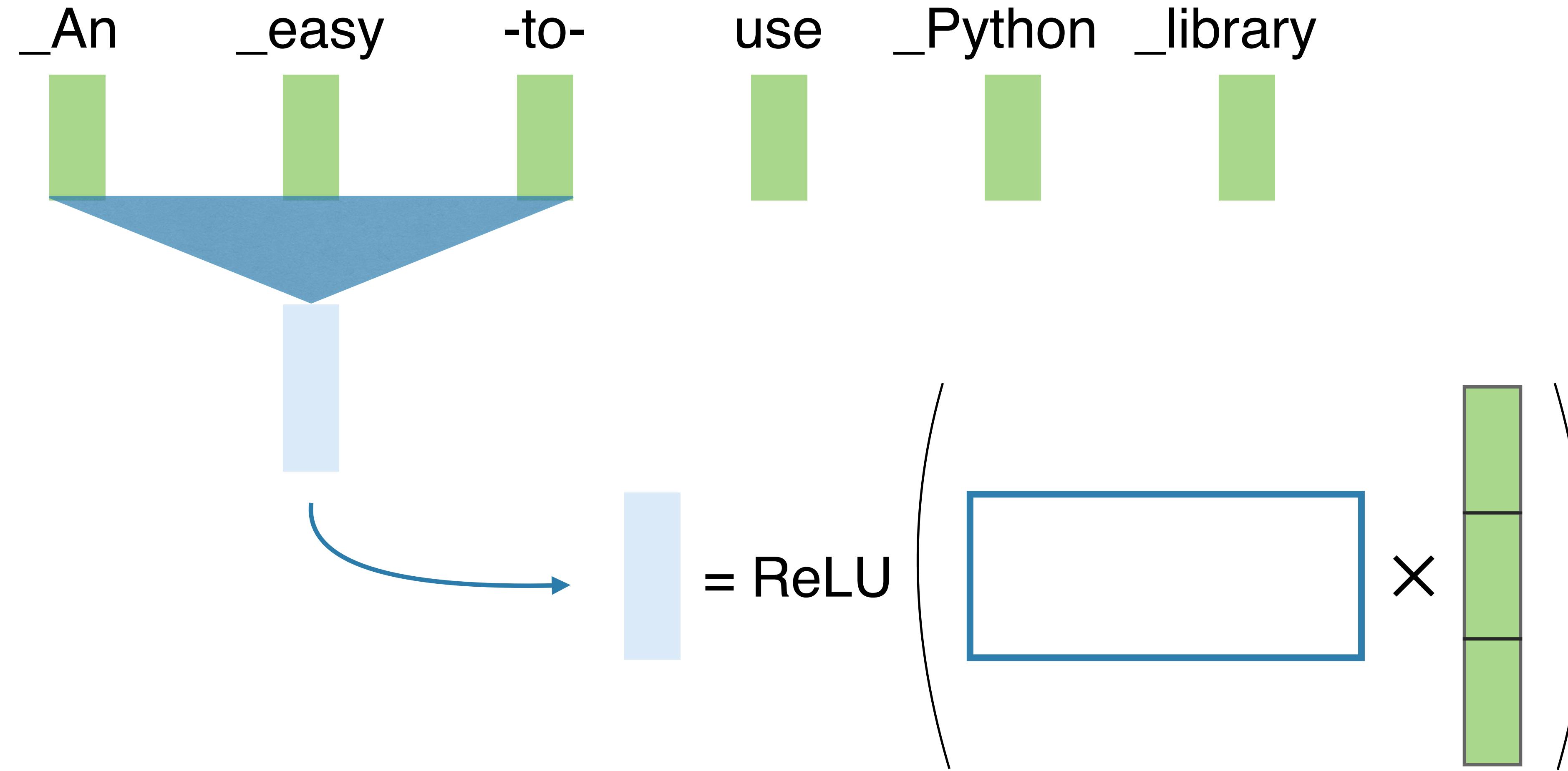
- RNN and CNN for NLP (in 10 minutes ...)
- Transformers architecture (continued):
 - Multi-head attention
 - Feed-forward layer
 - Residual connections
 - Normalization
 - Positional embeddings
- *Masked Language Modeling with transformers*
- Fine-tuning
- Scaling laws, emergent properties
- Issues, Open problems and questions

CNNs, RNNs

Neural networks for texts

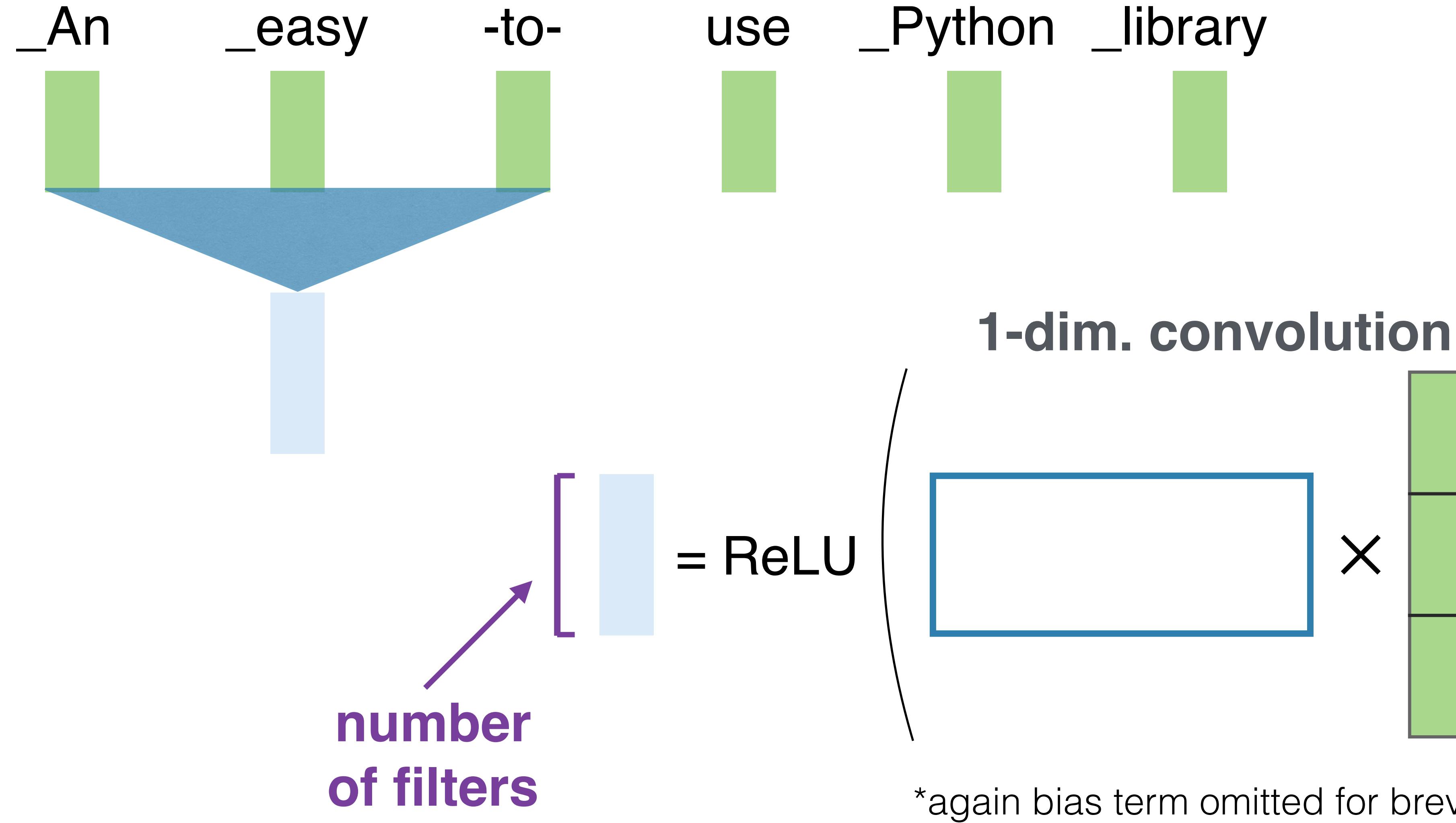


Convolutional neural networks for text (CNNs)

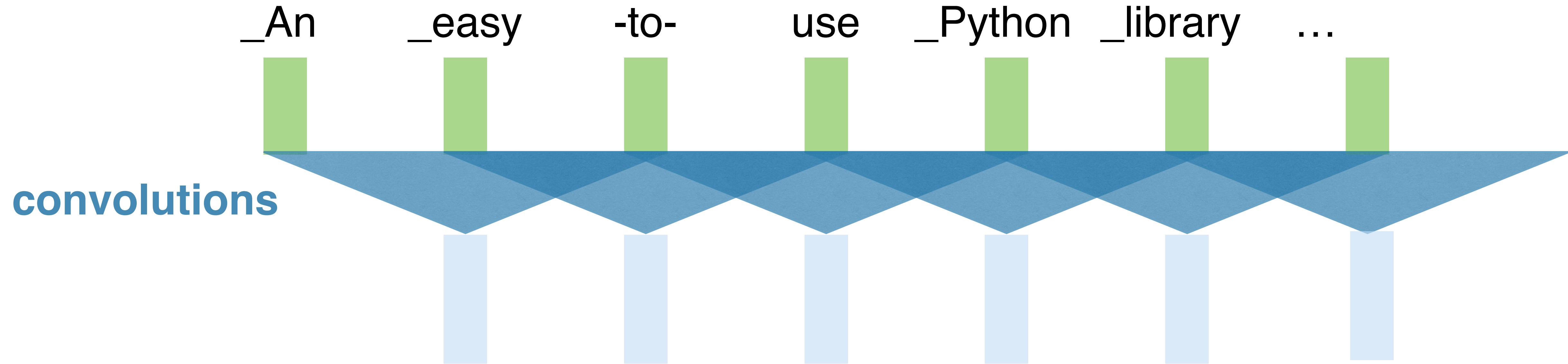


*again bias term omitted for brevity

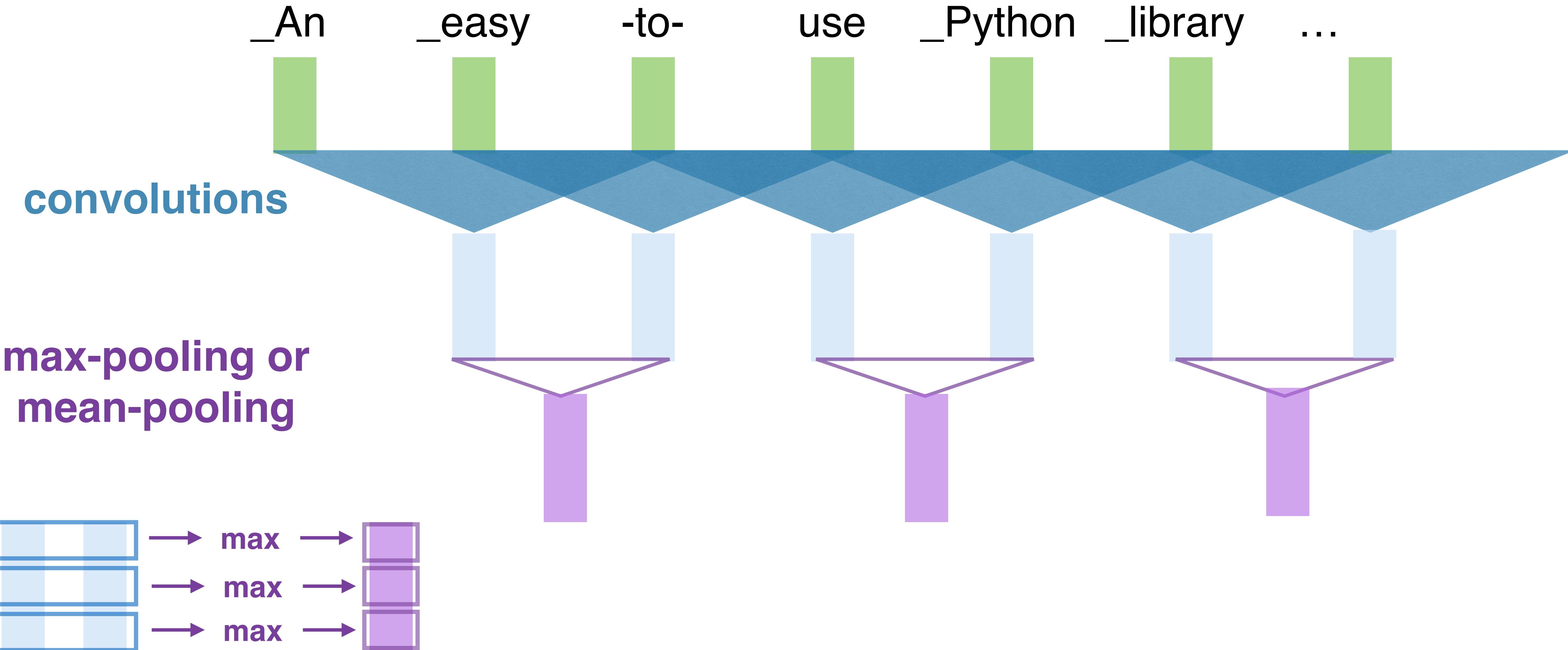
Convolutional neural networks for text (CNNs)



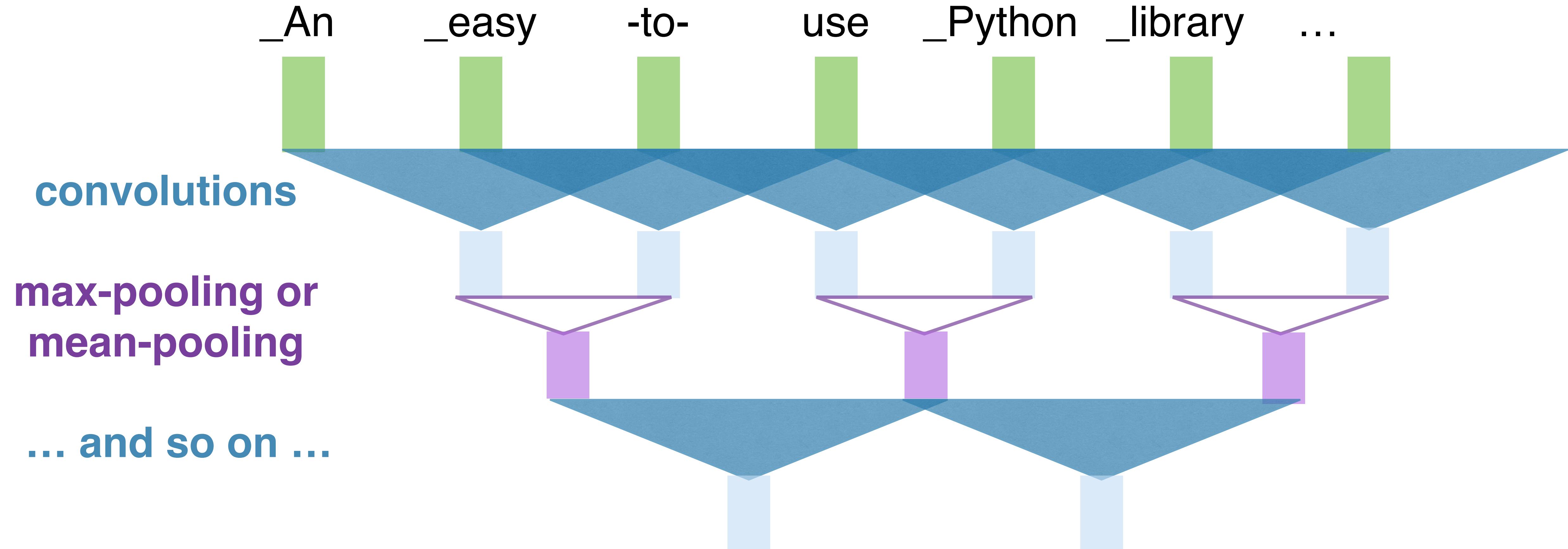
Convolutional neural networks for text (CNNs)



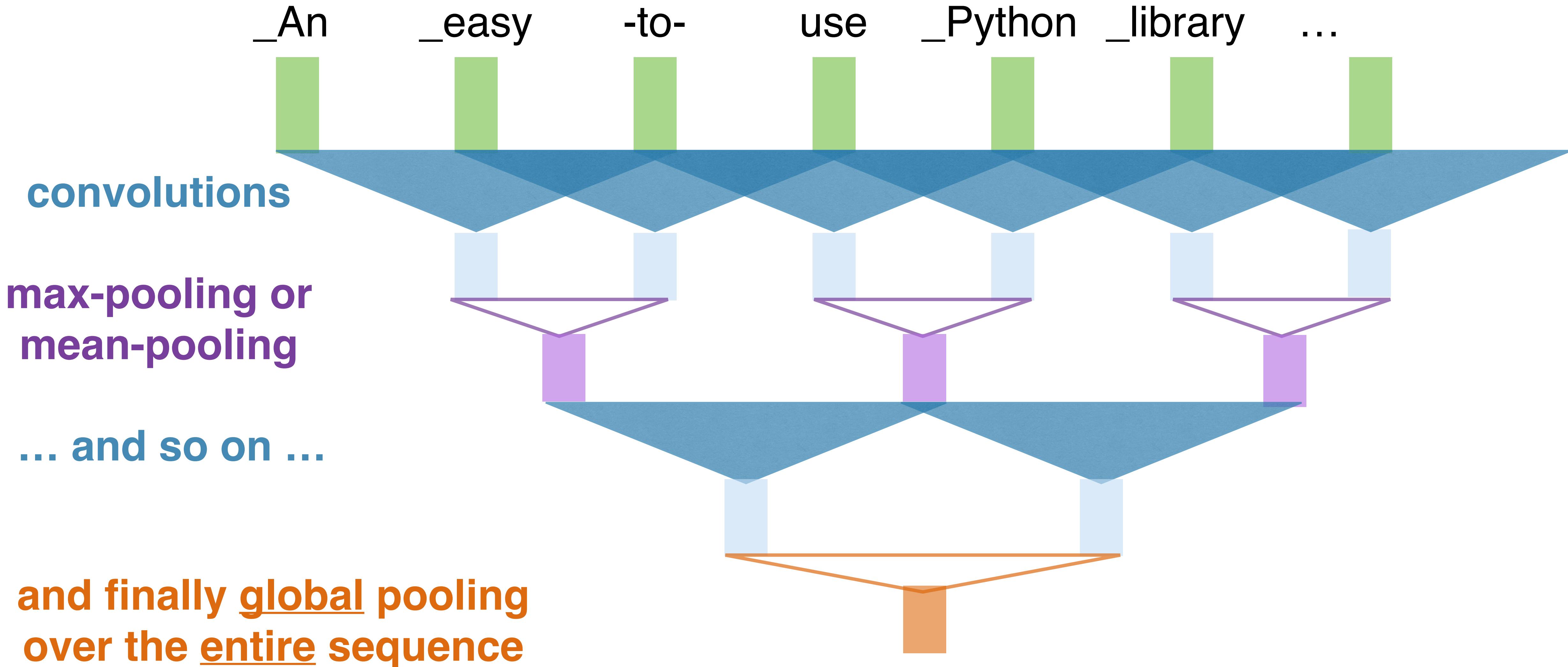
Convolutional neural networks for text (CNNs)



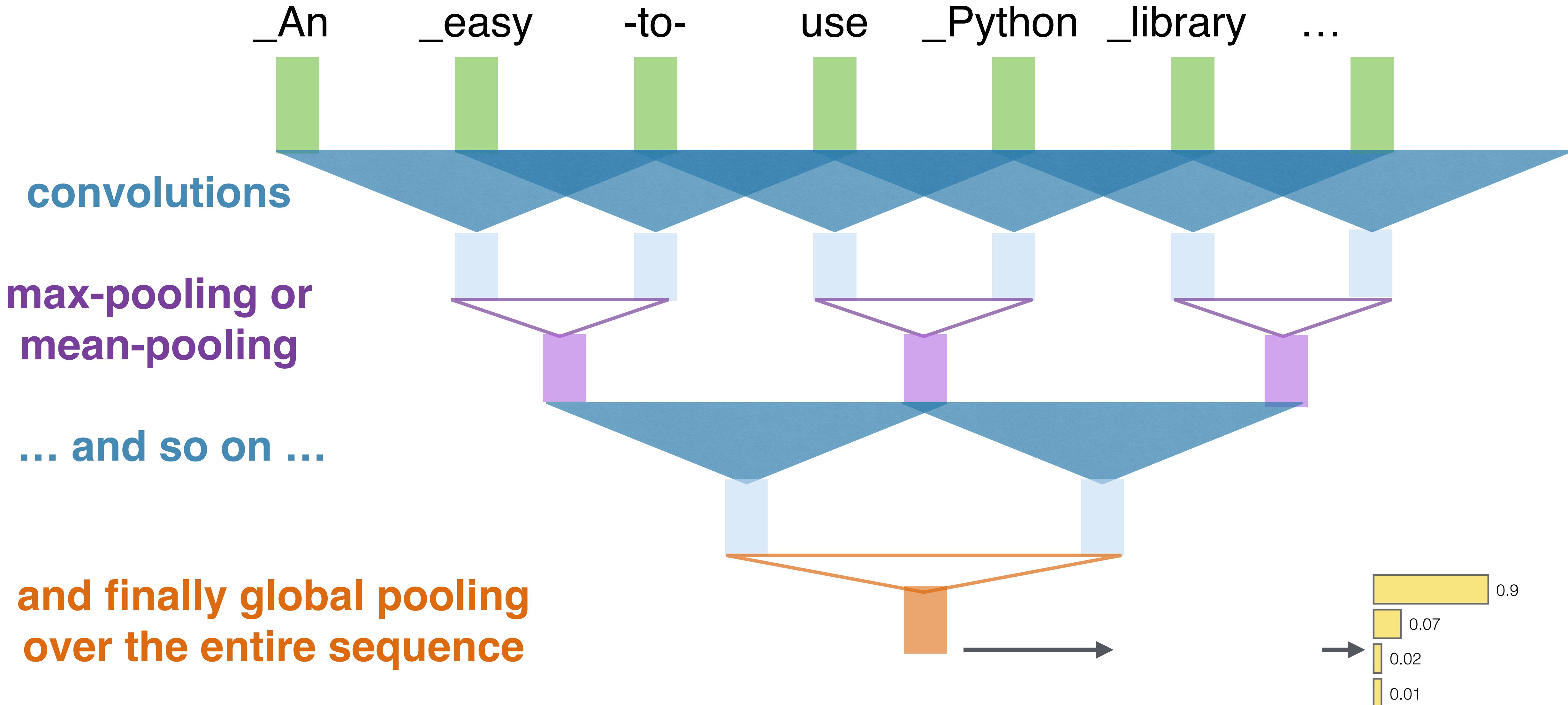
Convolutional neural networks for text (CNNs)



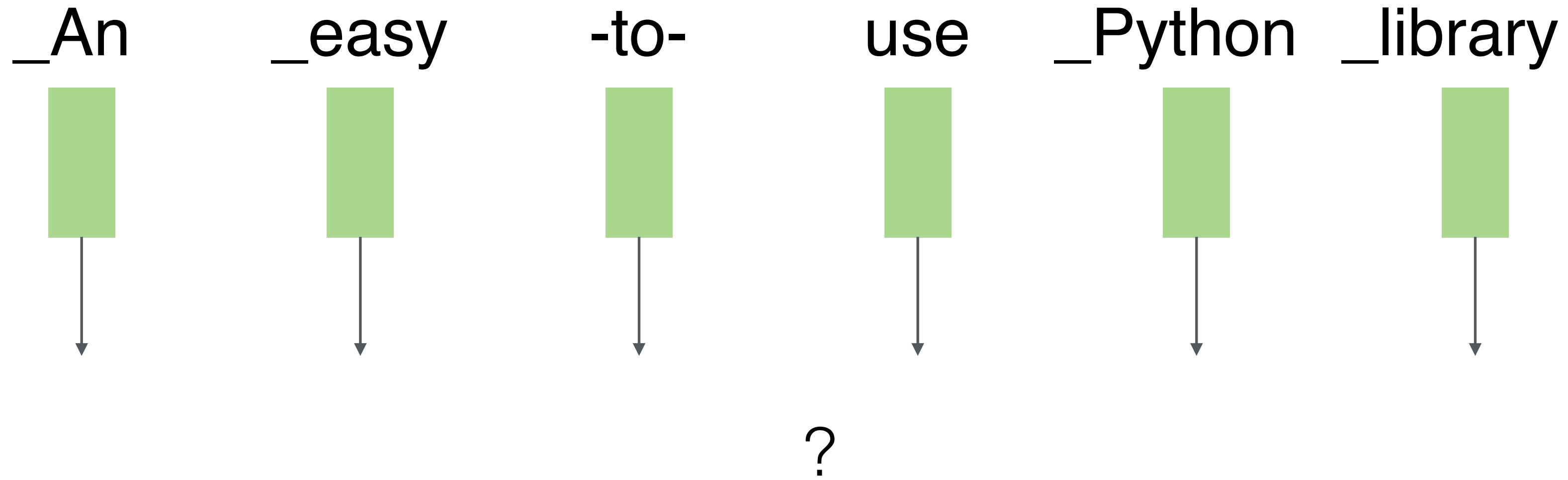
Convolutional neural networks for text (CNNs)



Convolutional neural networks for text (CNNs)



Neural networks for texts



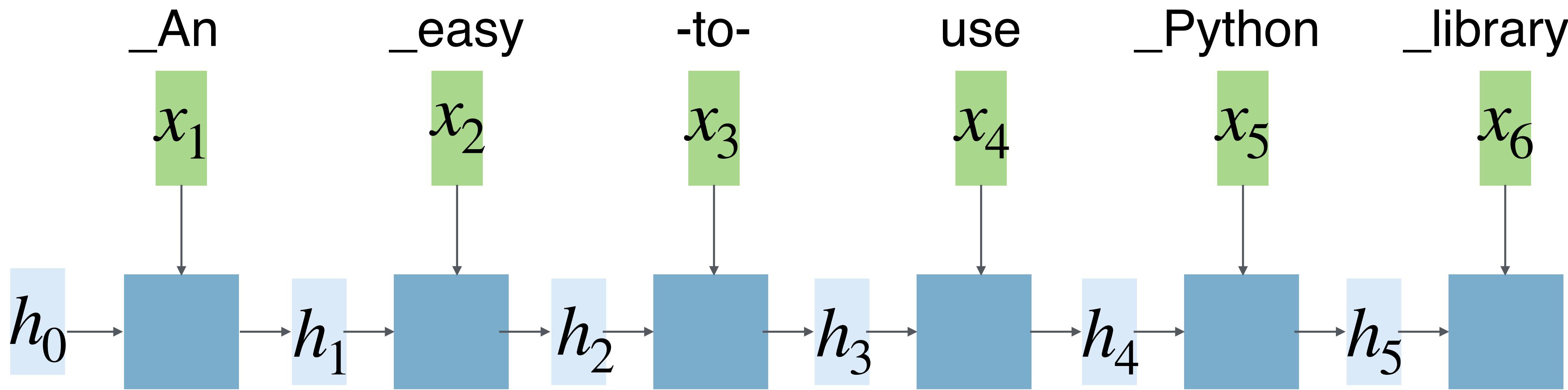
CNNs:

- + capture local context
- but long dependencies
are captured poorly

RNNs:

- + (in theory) capture various length context
- + reflect the sequential nature of text

Recurrent neural networks for texts (RNNs)



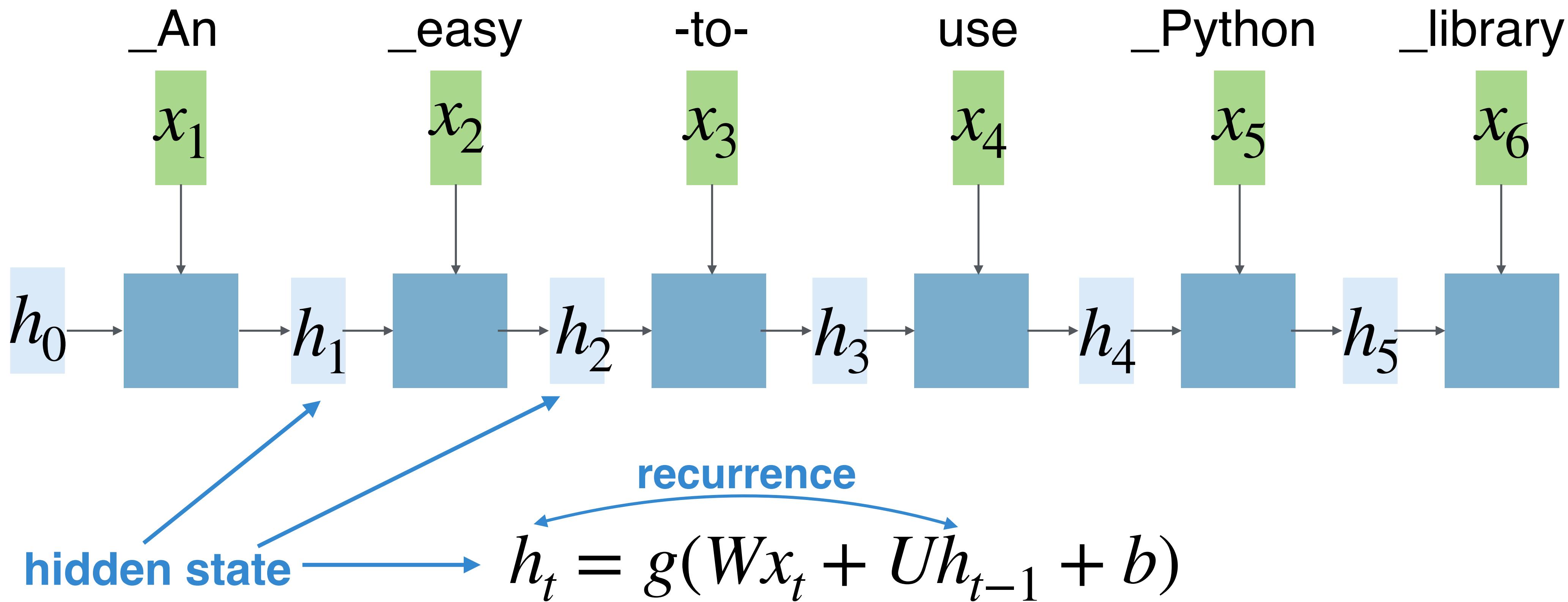
$$h_t = g(Wx_t + Uh_{t-1} + b)$$

$$h_t \in \mathbb{R}^m, \quad x_t \in \mathbb{R}^d, \quad W \in \mathbb{R}^{m \times d}, \quad U \in \mathbb{R}^{m \times m}, \quad b \in \mathbb{R}^m$$

e. g. $d = 512$,
 $m = 1024$

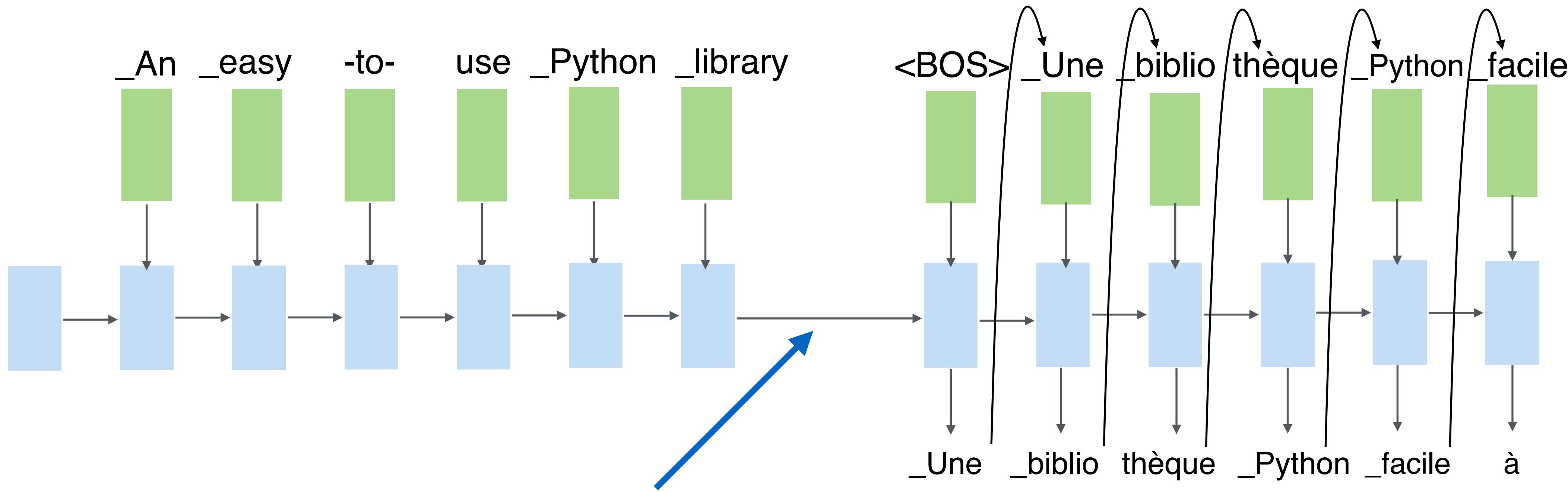
g is a nonlinearity function, e. g. \tanh
 $h_0 = 0$ or h_0 is a learned vector

Recurrent neural networks for texts (RNNs)



- same weights W , U , b are used at each time step
- reflect the sequential nature of text
- a very flexible model which can be used in various tasks

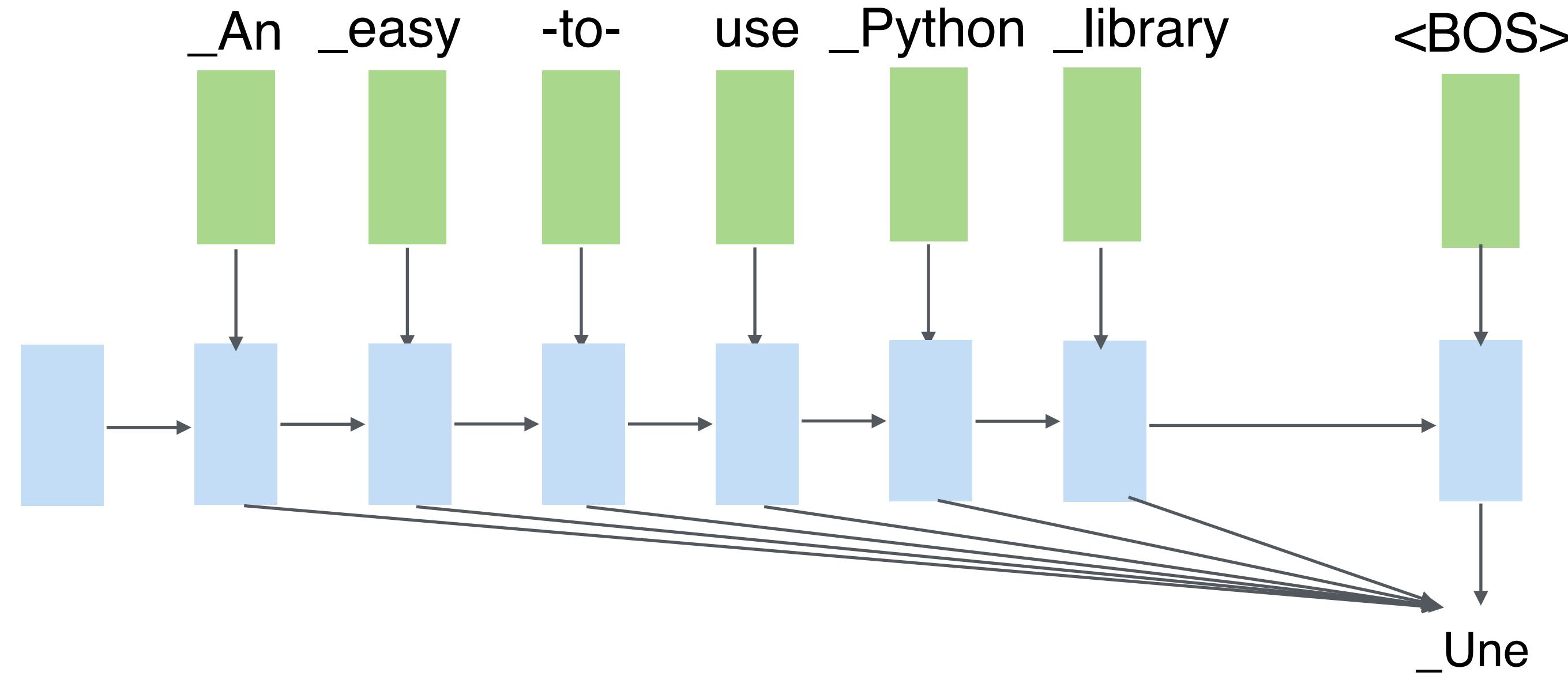
RNNs in sequence-to-sequence tasks



**Keeping all information about the source sequence
in the last hidden state is a bottleneck!**

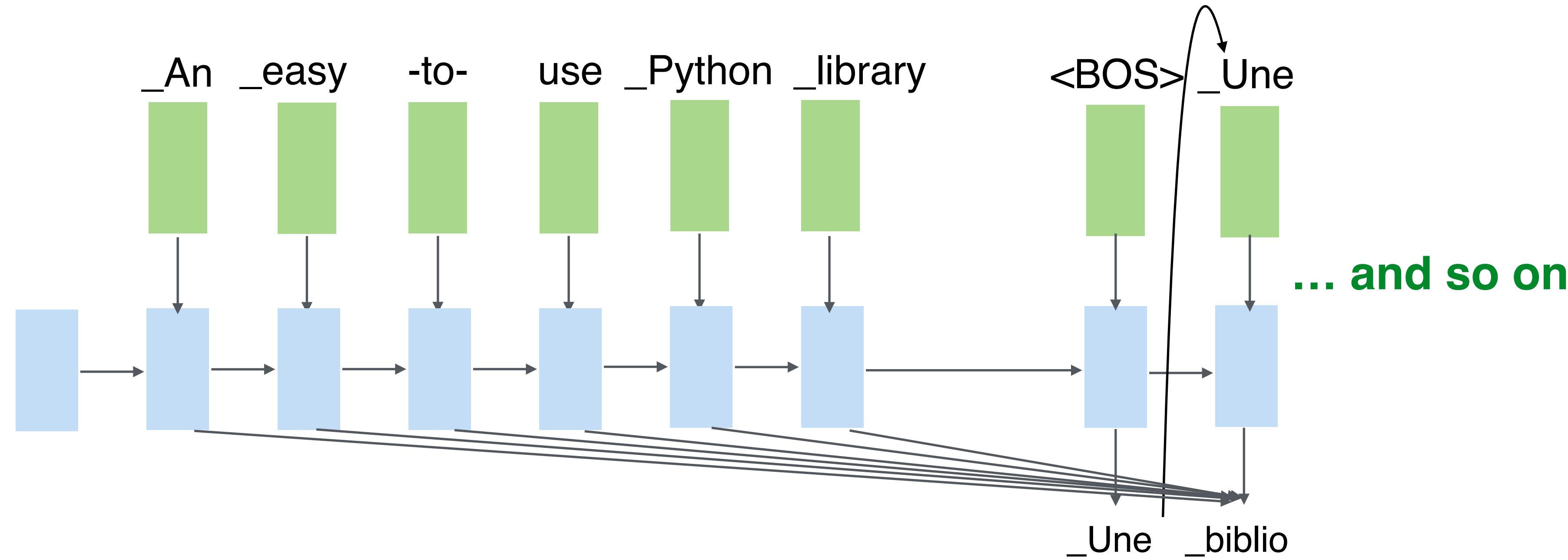
**Solution: “look” at the entire source sequence
when generating each target word**

Attention mechanism: basic idea



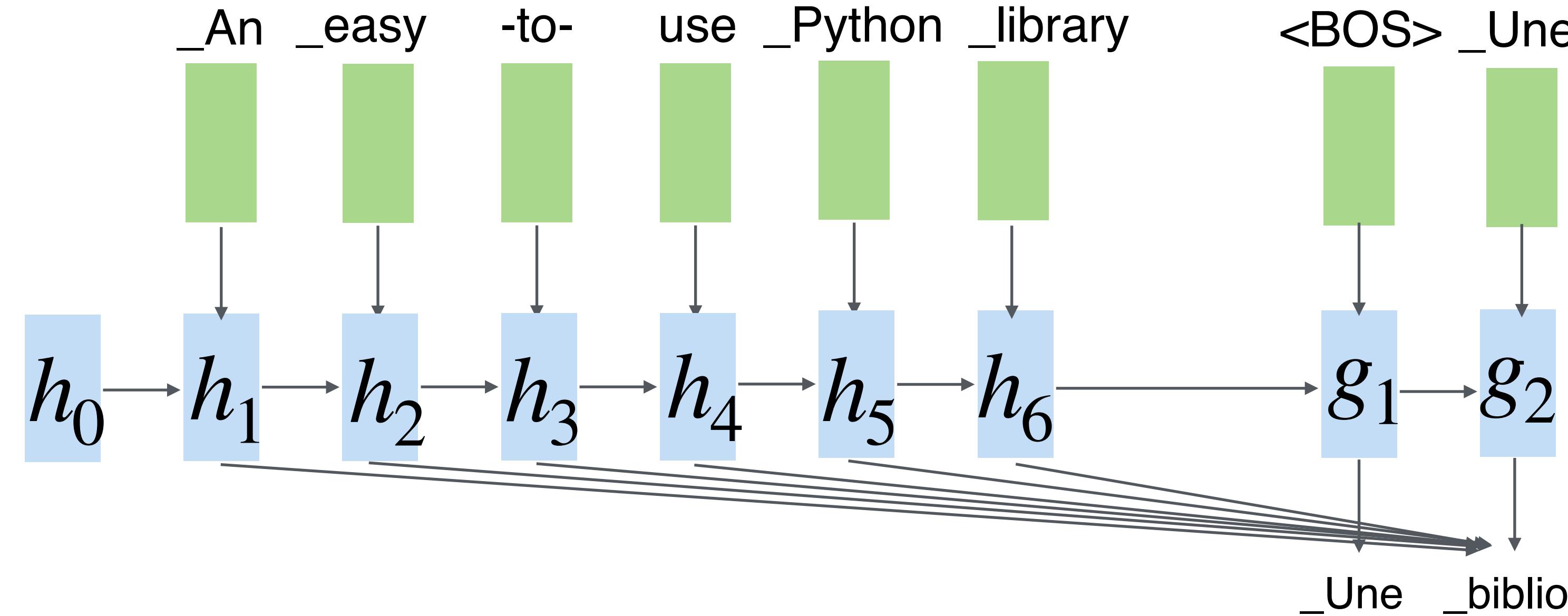
**Solution: “look” at the entire source sequence
when generating each target word**

Attention mechanism: basic idea



**Solution: “look” at the entire source sequence
when generating each target word**

Attention mechanism: example formulas



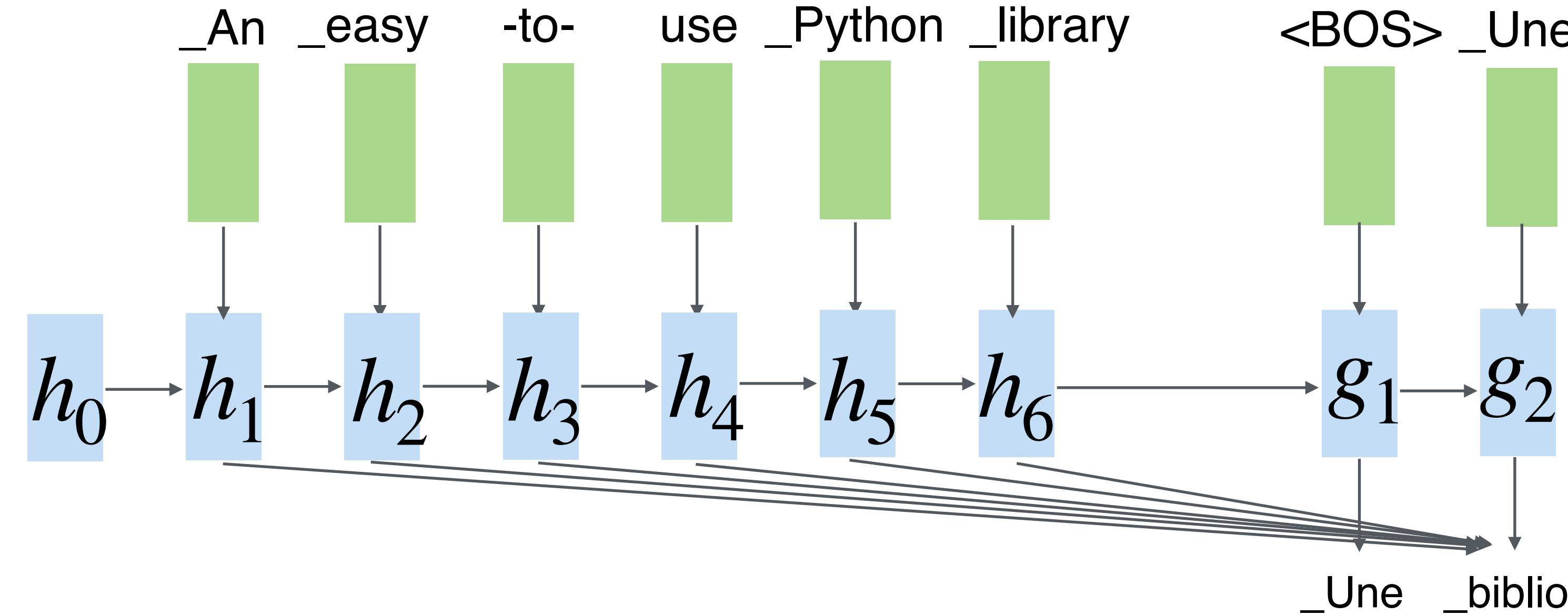
- Step 1:

$$\alpha_{kt} = \frac{\exp(h_k^T g_t)}{\sum_{k'=1}^T \exp(h_{k'}^T g_t)}$$

**“relevance”
of source token t
when generating
target token k**

- dot product to measure “relevance”
- softmax to achieve categorical distribution (sum=1)
- T is the length of the input sequence

Attention mechanism: example formulas



- Step 1:

$$\alpha_{kt} = \frac{\exp(h_k^T g_t)}{\sum_{k'=1}^T \exp(h_{k'}^T g_t)}$$

**“relevance”
of source token t
when generating
target token k**

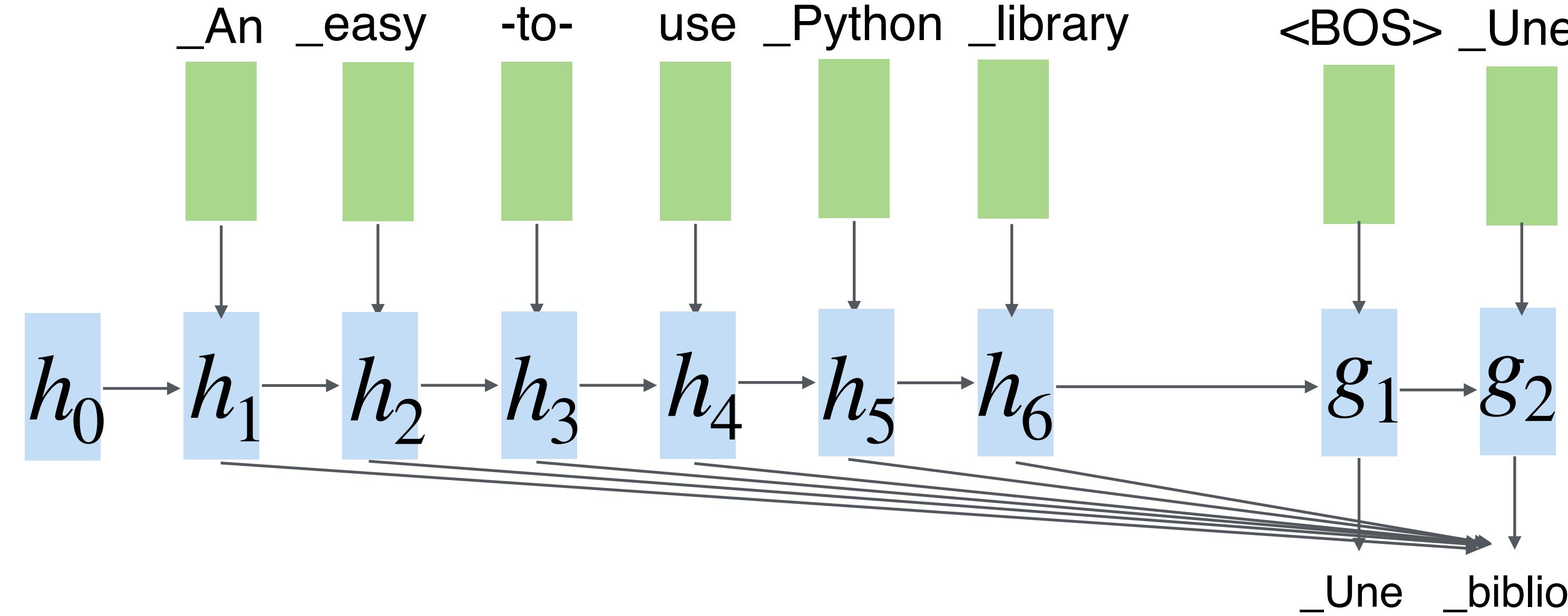
- Step 2:

$$c_t = \sum_{k=1}^T \alpha_{kt} h_k$$

**context vector is a weighted
sum of source hidden states**

- dot product to measure “relevance”
- softmax to achieve categorical distribution (sum=1)
- T is the length of the input sequence

Attention mechanism: example formulas



- Step 1:

$$\alpha_{kt} = \frac{\exp(h_k^T g_t)}{\sum_{k'=1}^T \exp(h_{k'}^T g_t)}$$

**“relevance”
of source token t
when generating
target token k**

- dot product to measure “relevance”
- softmax to achieve categorical distribution (sum=1)
- T is the length of the input sequence

- Step 2:

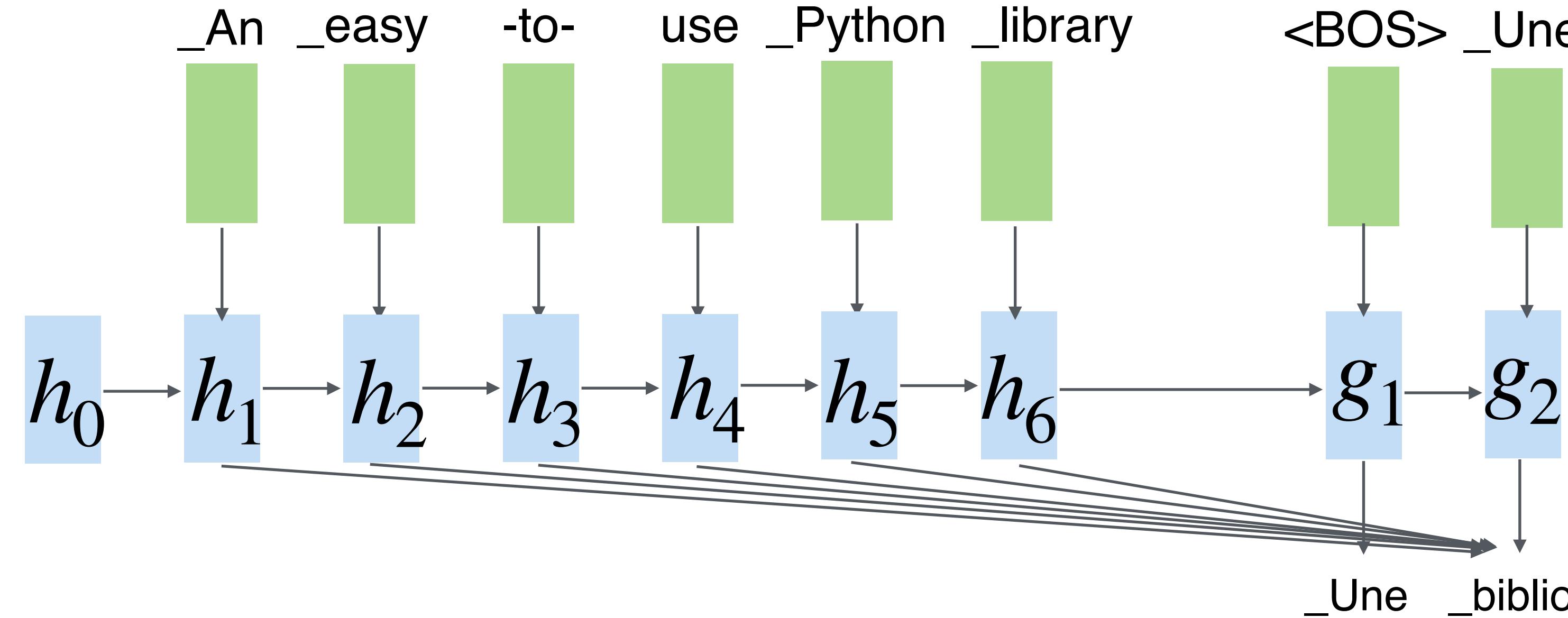
$$c_t = \sum_{k=1}^T \alpha_{kt} h_k$$

**context vector is a weighted
sum of source hidden states**

- Step 3:

Context vector c_t is used along with hidden state g_t in a linear layer to generate the next token

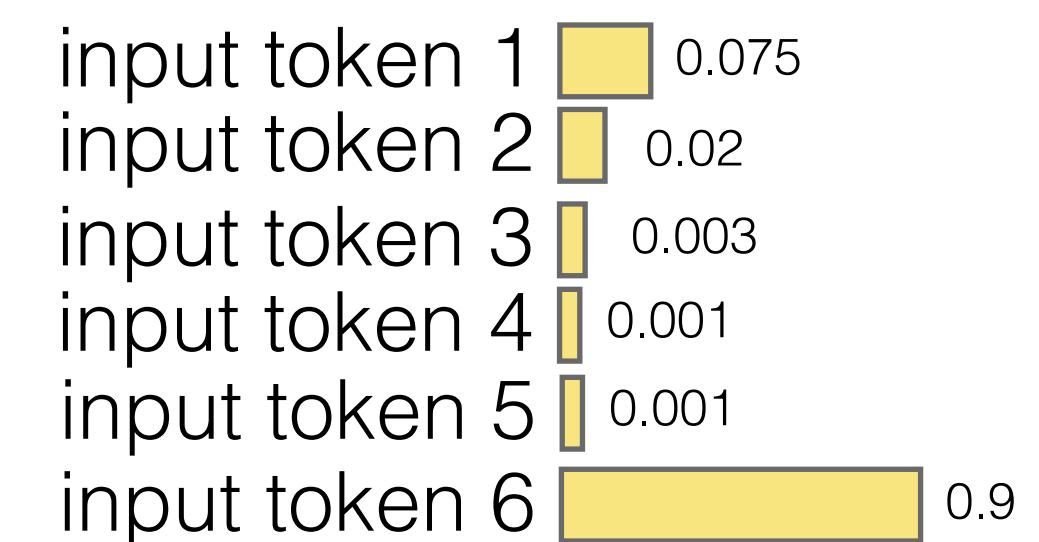
Attention mechanism: example formulas



For example, when generating token “_biblio” at step $k = 2$,
the model may produce the following attention weights:

$$\alpha_{kt} = \frac{\exp(h_k^T g_t)}{\sum_{k'=1}^T \exp(h_{k'}^T g_t)}$$

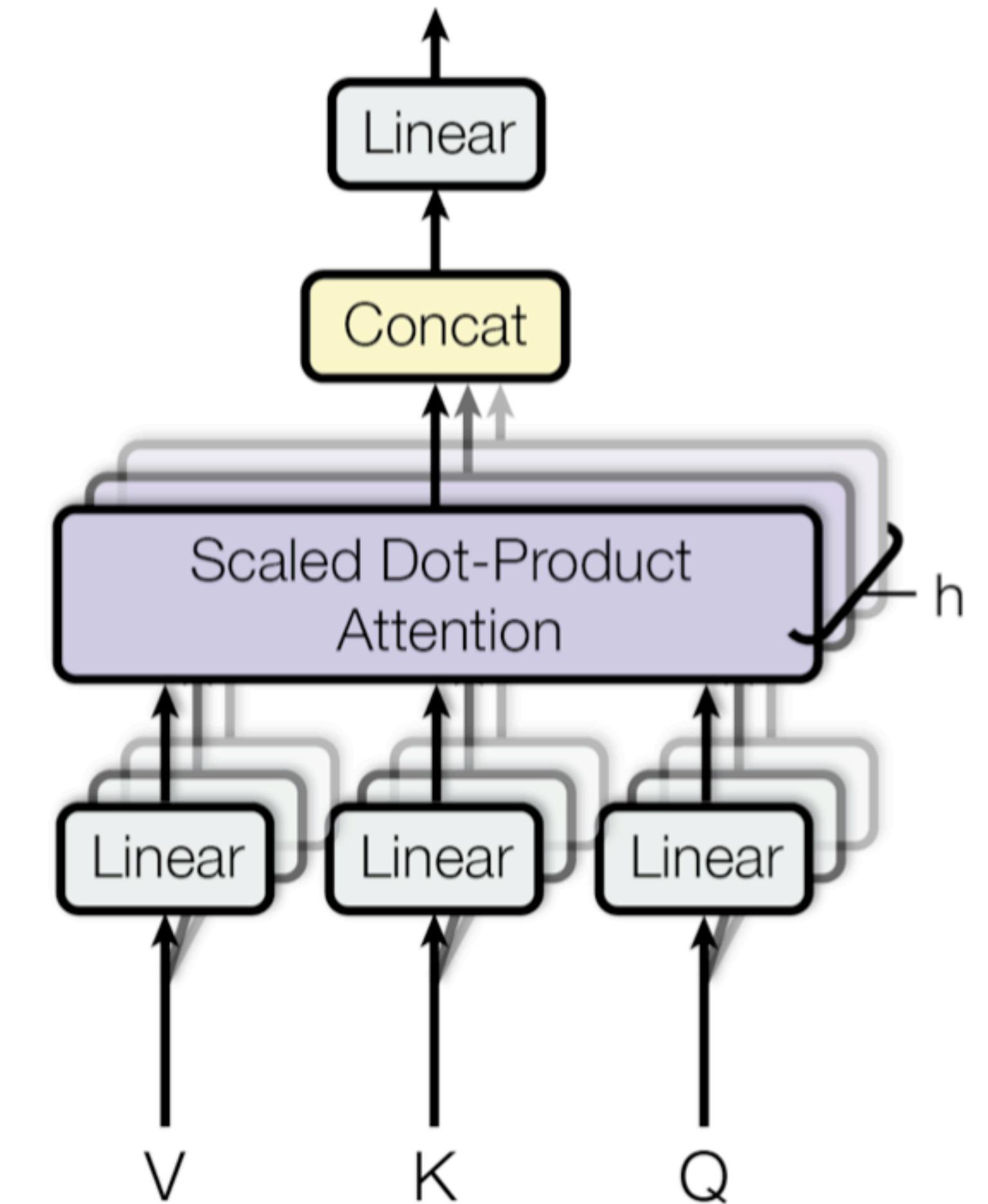
$$\begin{array}{lll} \alpha_{12} = 0.075 & \alpha_{22} = 0.02 & \alpha_{32} = 0.003 \\ \text{_An} & \text{_easy} & \text{-to-} \\ \alpha_{42} = 0.001 & \alpha_{52} = 0.001 & \alpha_{62} = 0.9 \\ \text{use} & \text{_Python} & \text{_library} \end{array}$$



Transformers architecture (Continued)

Multi-head attention

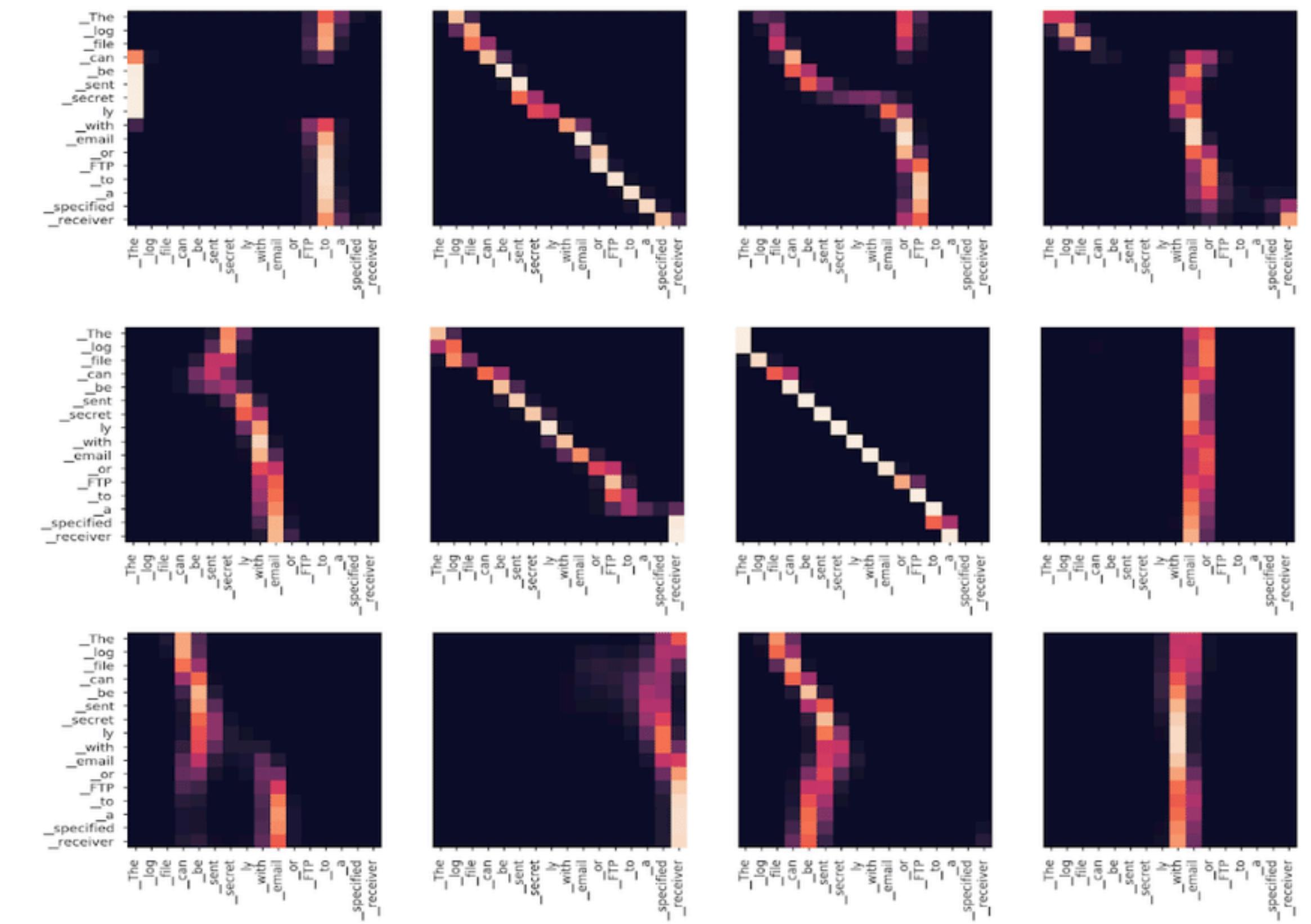
- In practice, multiple attentions *heads* are computed in parallel and gathered
- Their individual outputs are concatenated and a last linear layer (the “up” projection, is applied)
- It’s an important part of transformer success



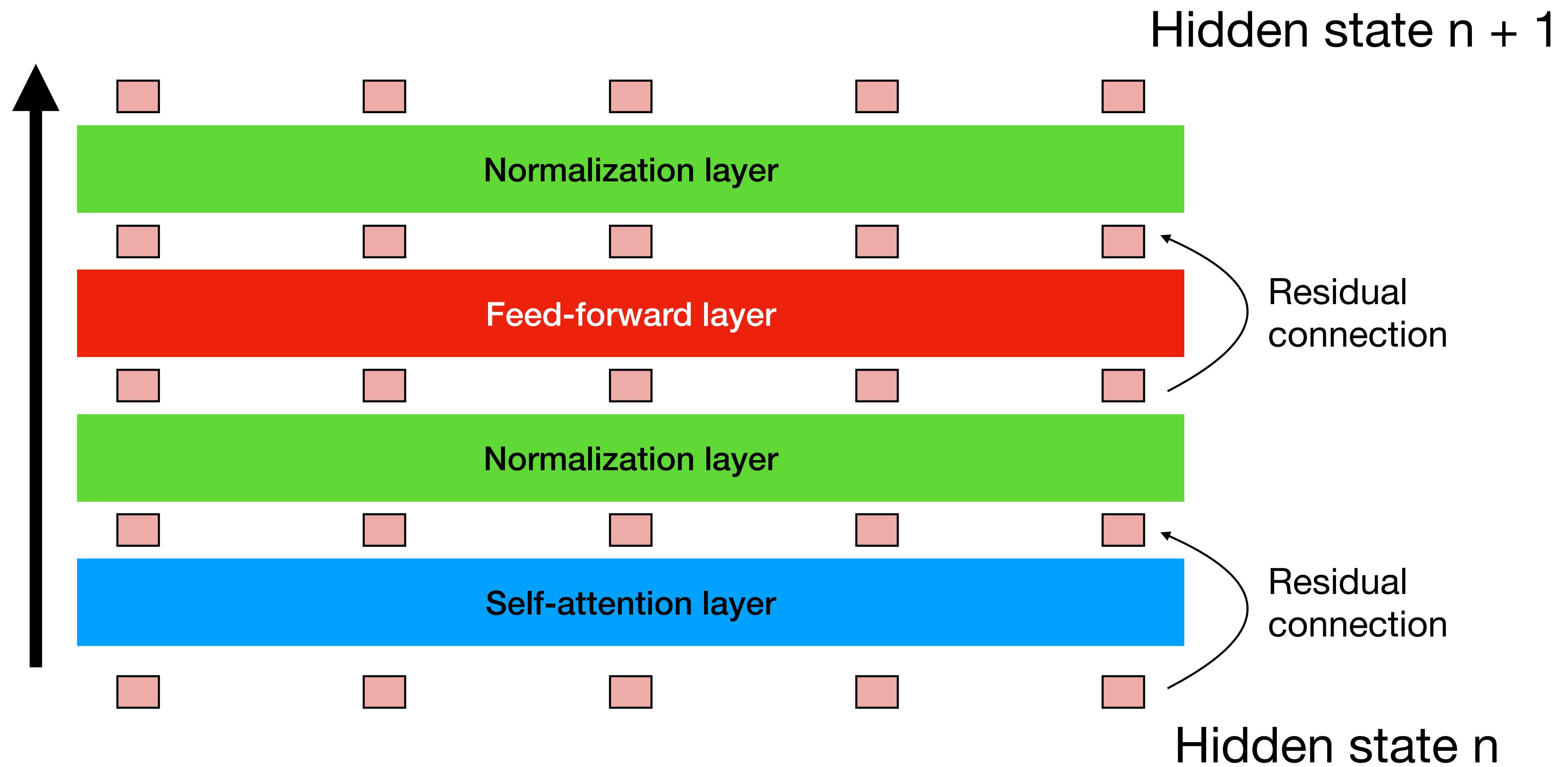
Here, each V, K, Q has dimension $\frac{n_{\text{emb}}}{n_{\text{head}}}$

Multi-head attention

- It increases the expressiveness (the network is free to learn to formulate multiple ‘query’ domains in attention)...
- ... in a parallelizable fashion: each head can be computed separately
- <https://bbycroft.net/lm>



Inside a Transformer block



Feed-forward layer

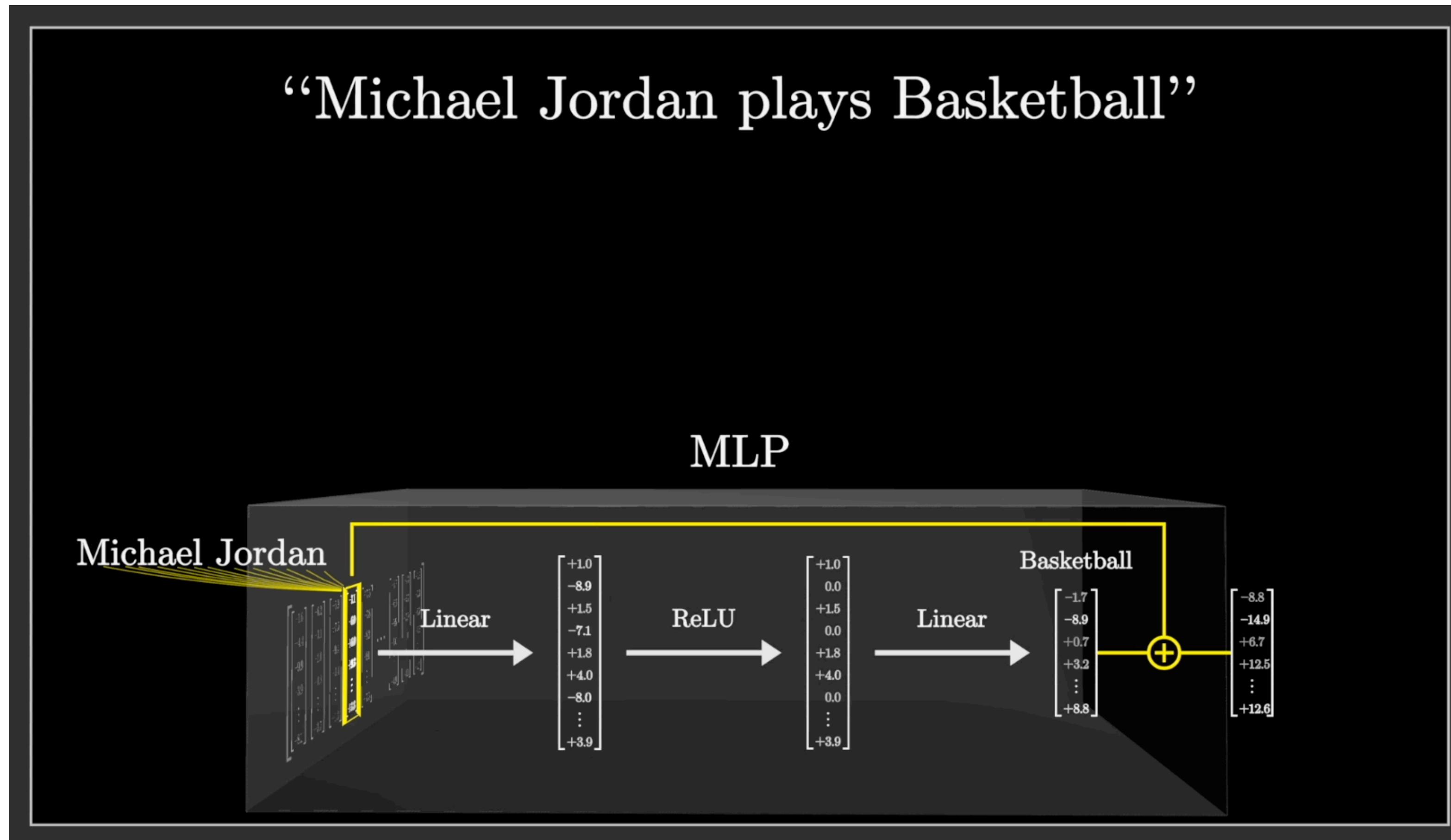
Two fully connected layers

$$\text{MLP}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

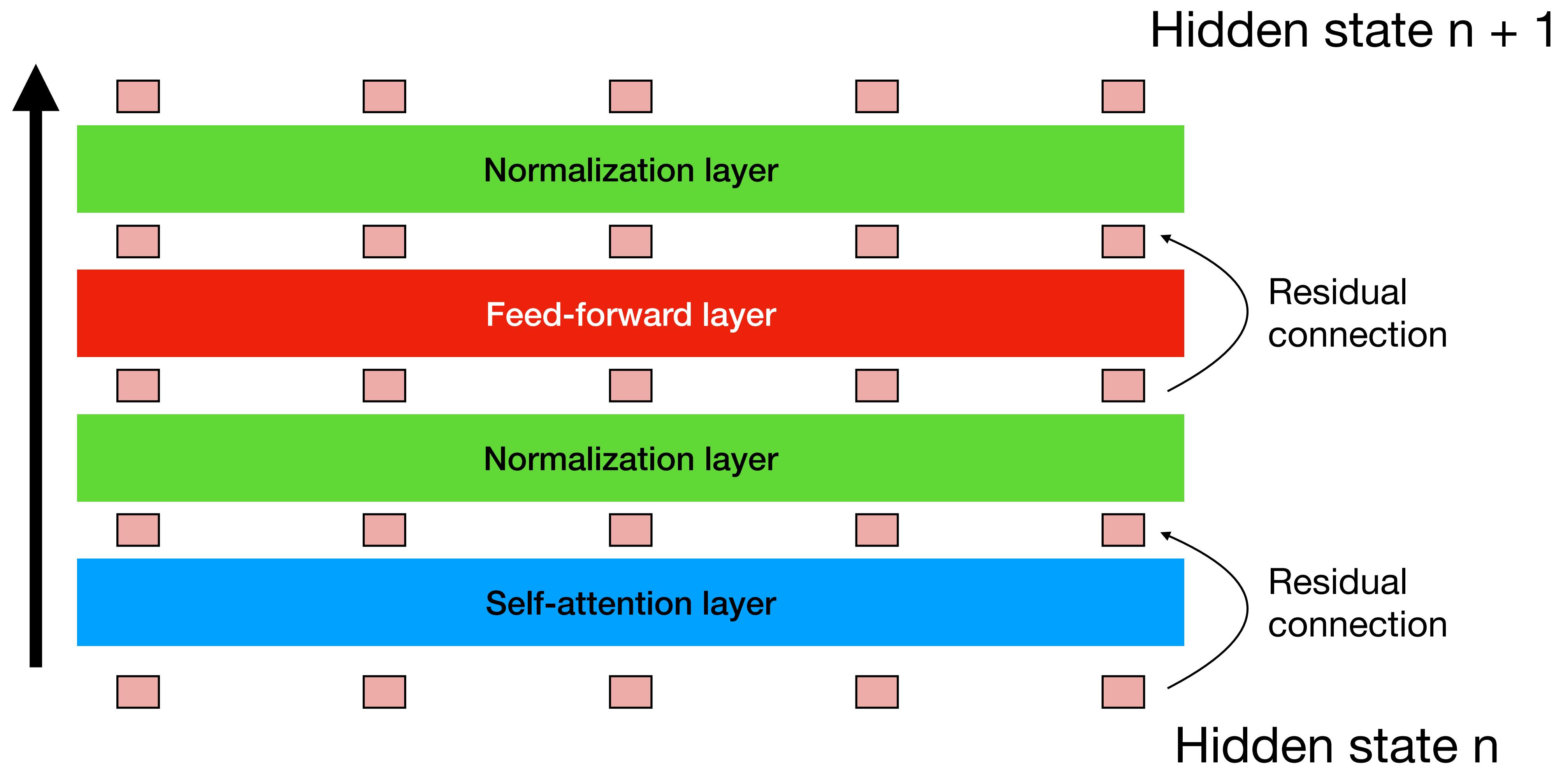
- Applied token-wise
- It can be viewed (arguably) as the ‘memory’ of the transformers.

Feed-forward layer

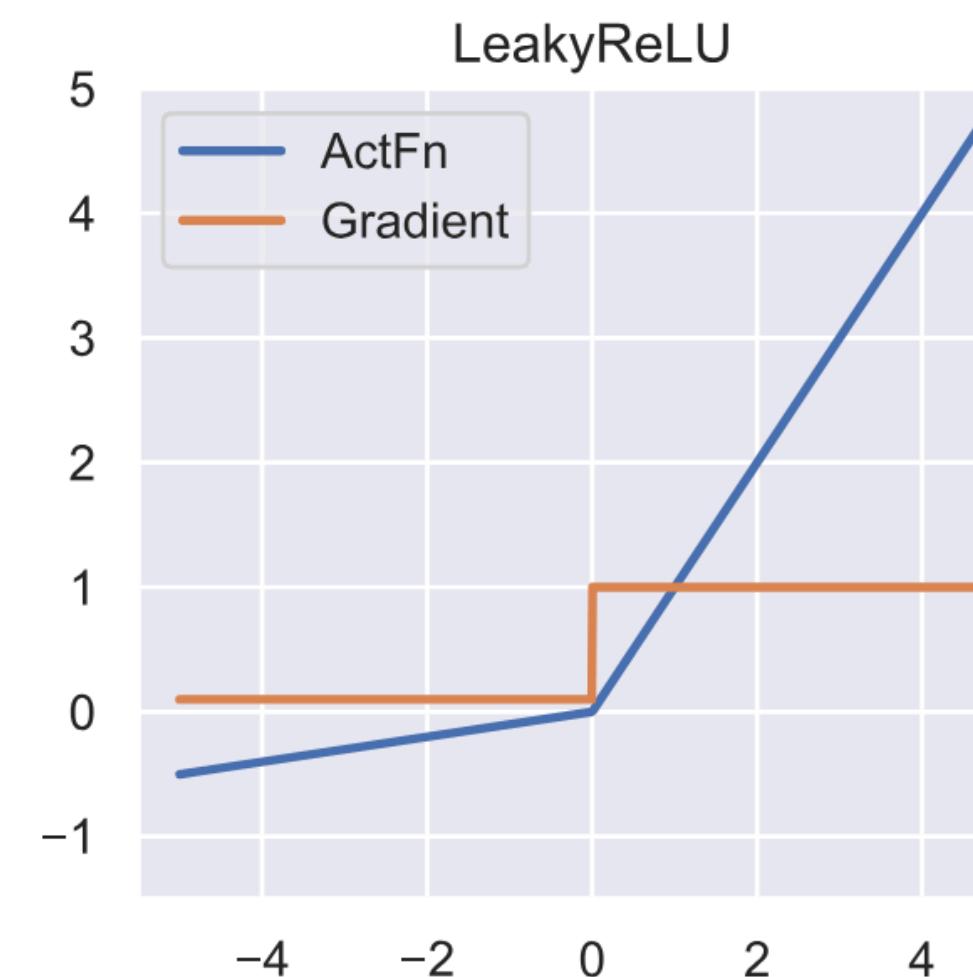
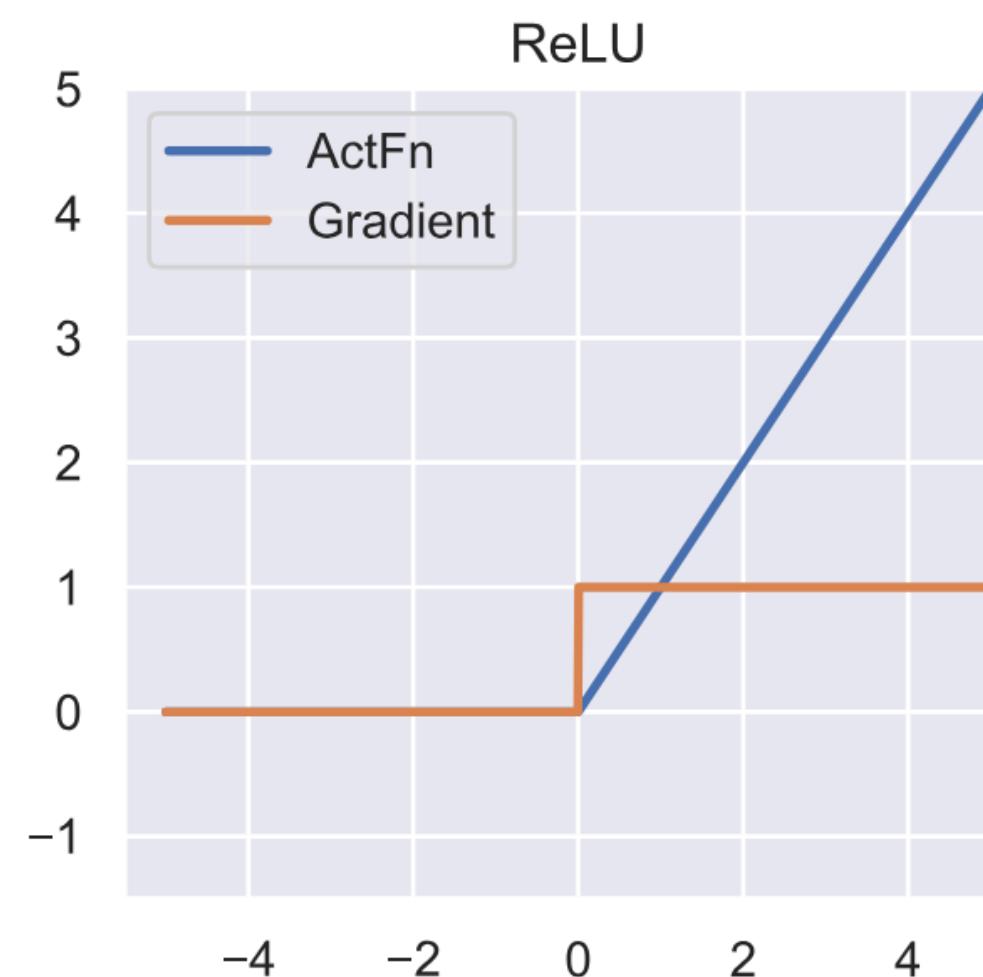
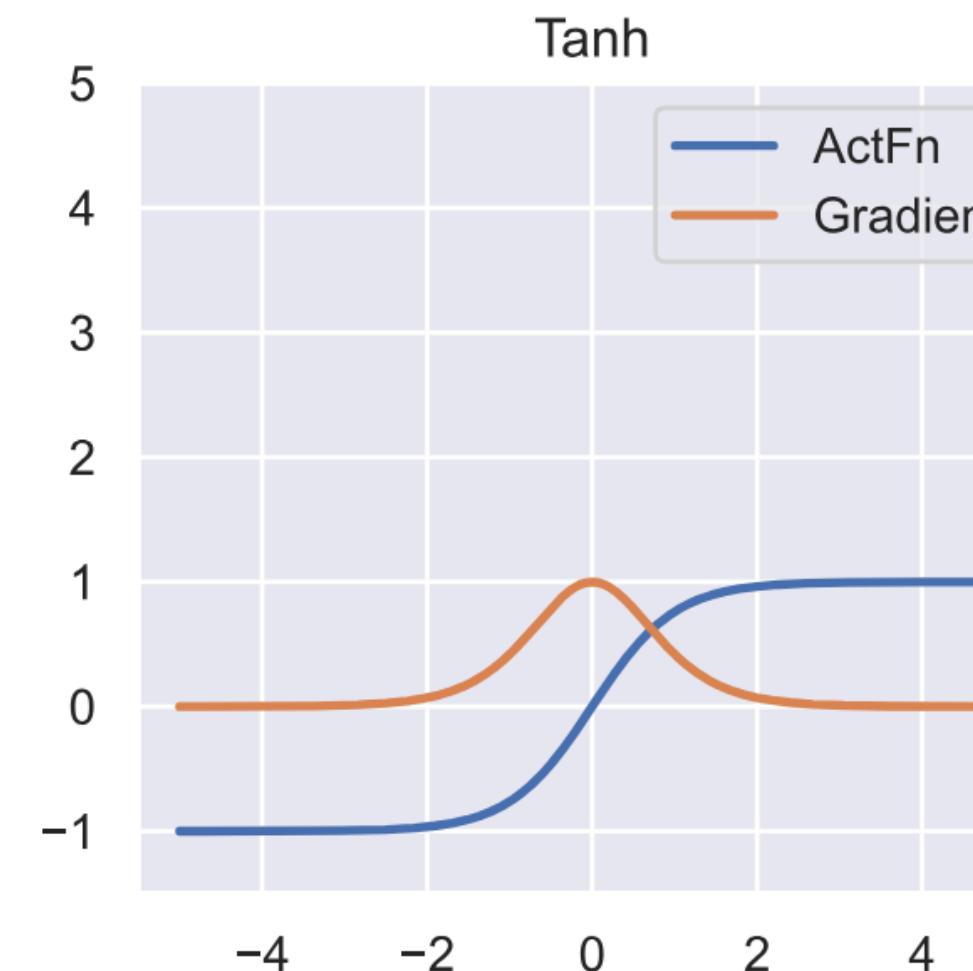
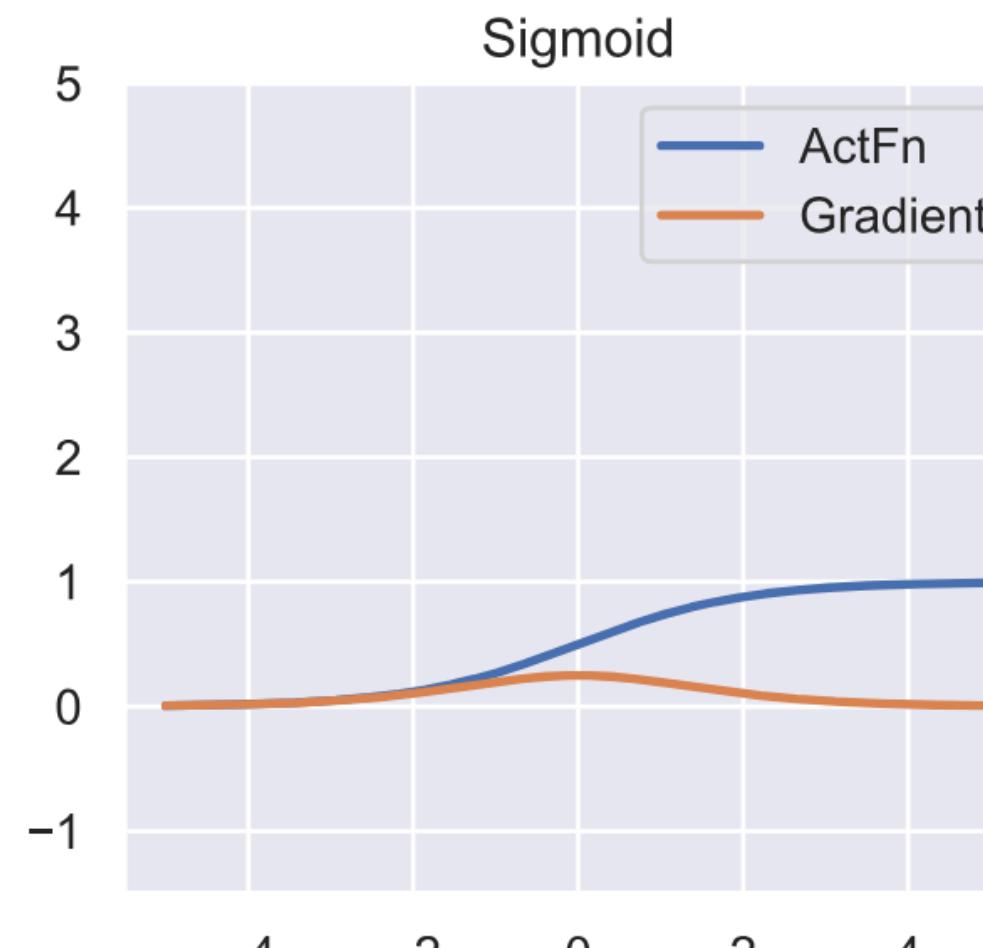
Two fully connected layers



Inside a Transformer block



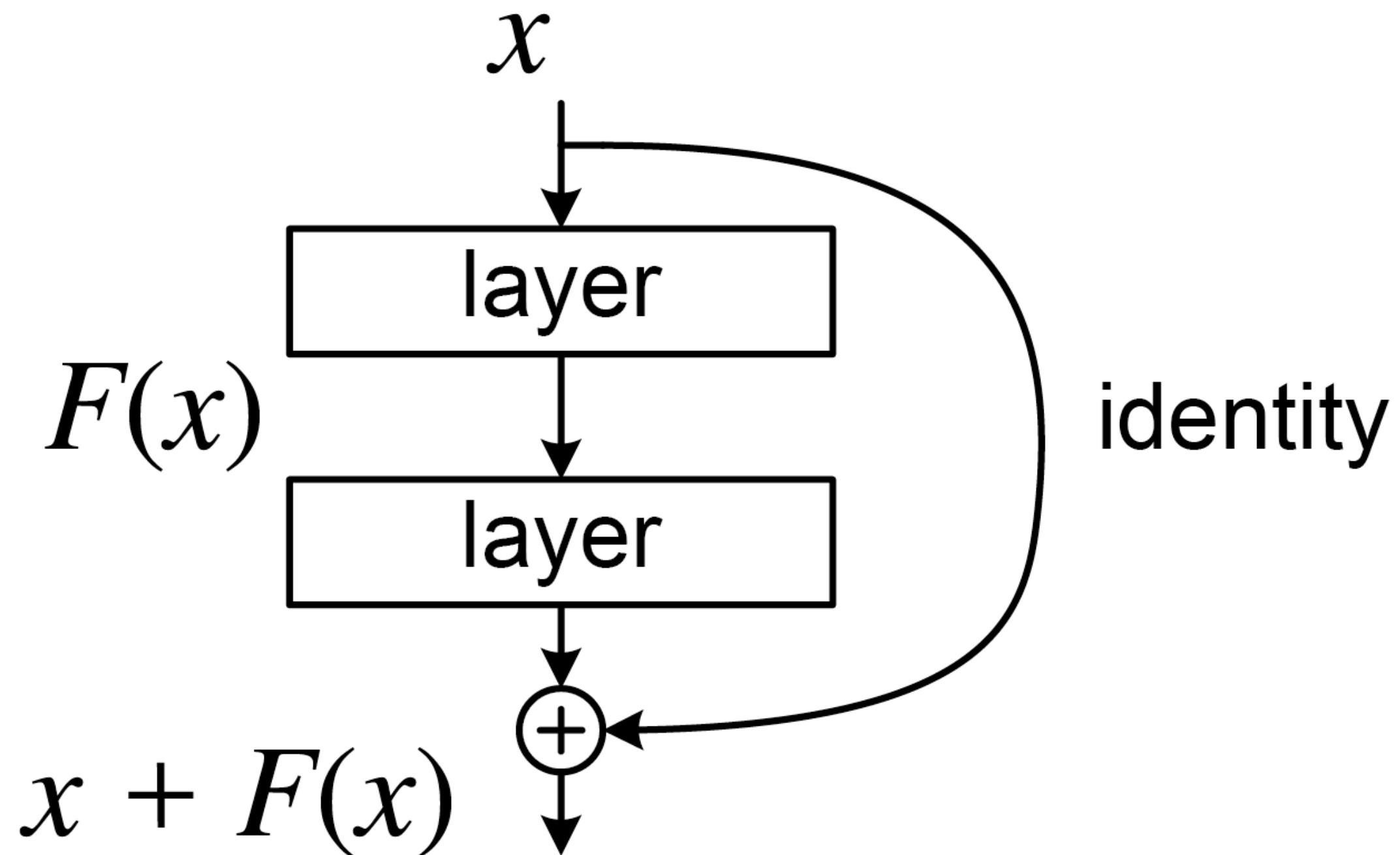
The problem of vanishing gradients



- Iterative back propagation with DL activations leads to vanishing gradients
- Learning **very deep** neural networks with iterative layer is challenging
- One solution: residual connection

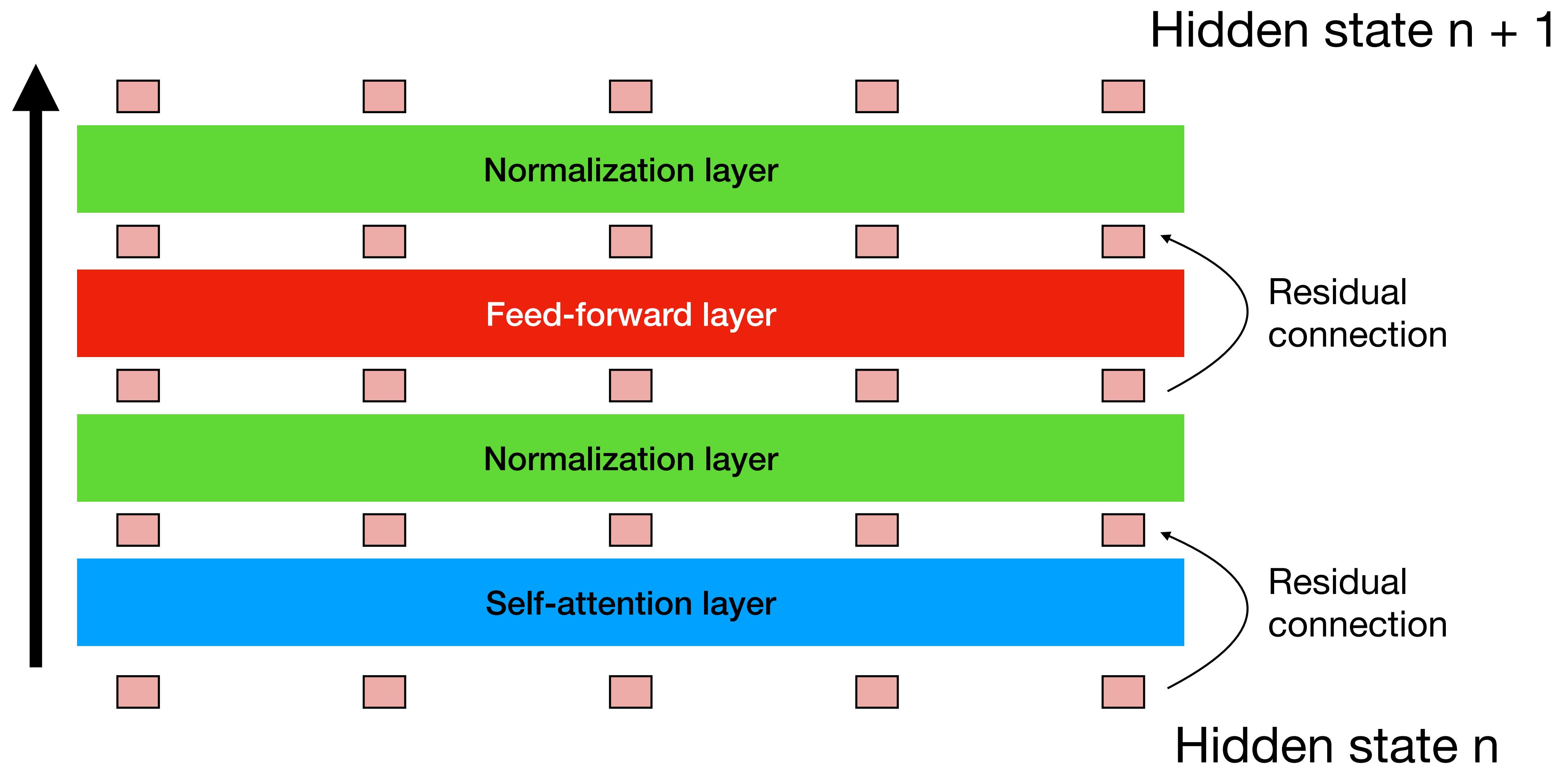
Residual connection

Avoiding vanishing gradients



- Add output of N-th layer to output of N+1-th layer
- Gradient computation involves the layer as before + the gradient through the identity: **no vanishing**
- BUT Different layers activations must match in dimension !
- **All large neural networks have residual connections.**

Inside a Transformer block



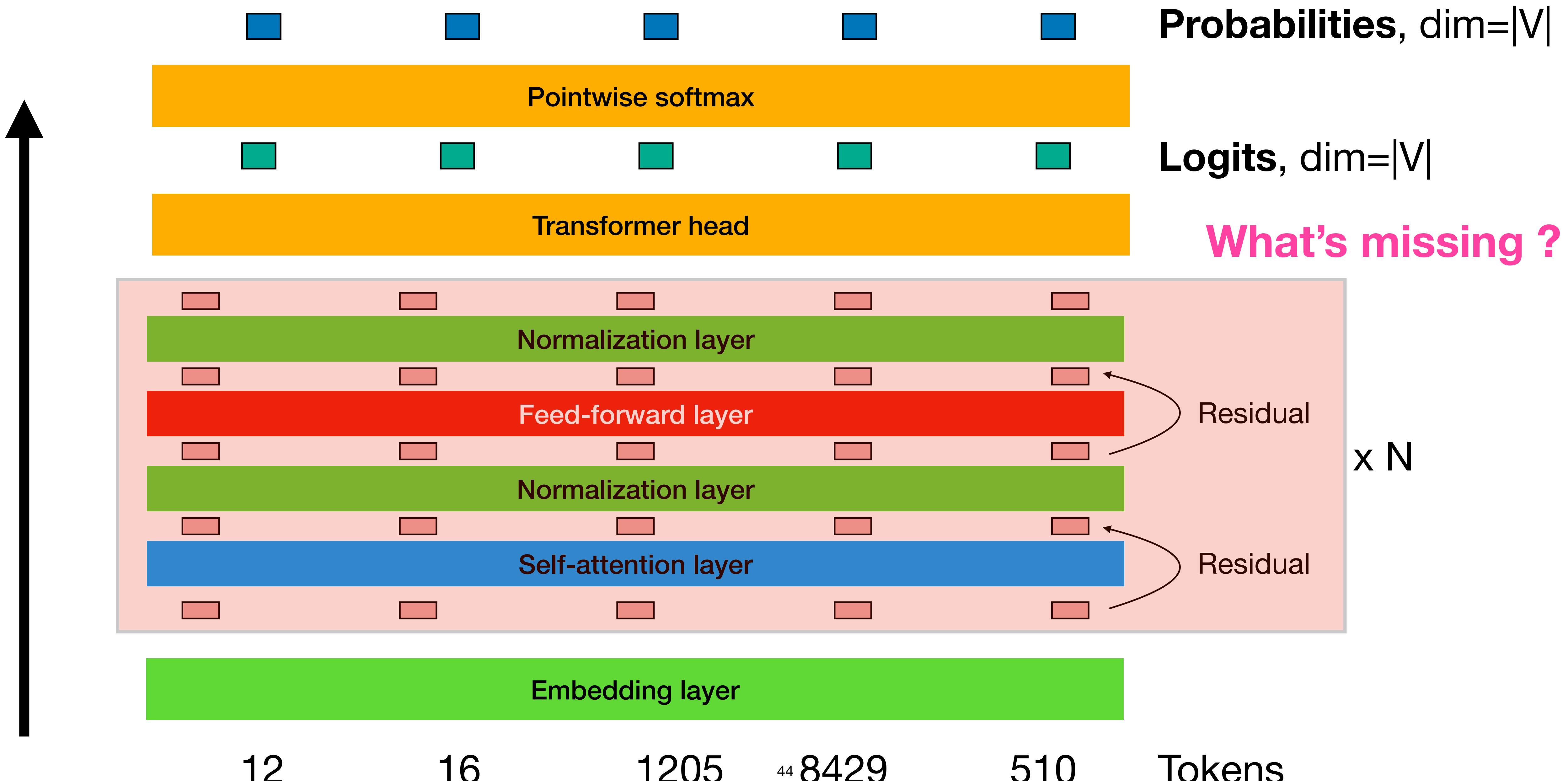
Normalization layer

- **Internal covariate shift** causes optimization instabilities for large deep neural networks (variance of each layer input activation scales across batches through optimization)
- Therefore, a normalization of the activations (i.e. hidden states for a transformers) improves the learning procedure.
- Transformers typically use Layer Normalization:

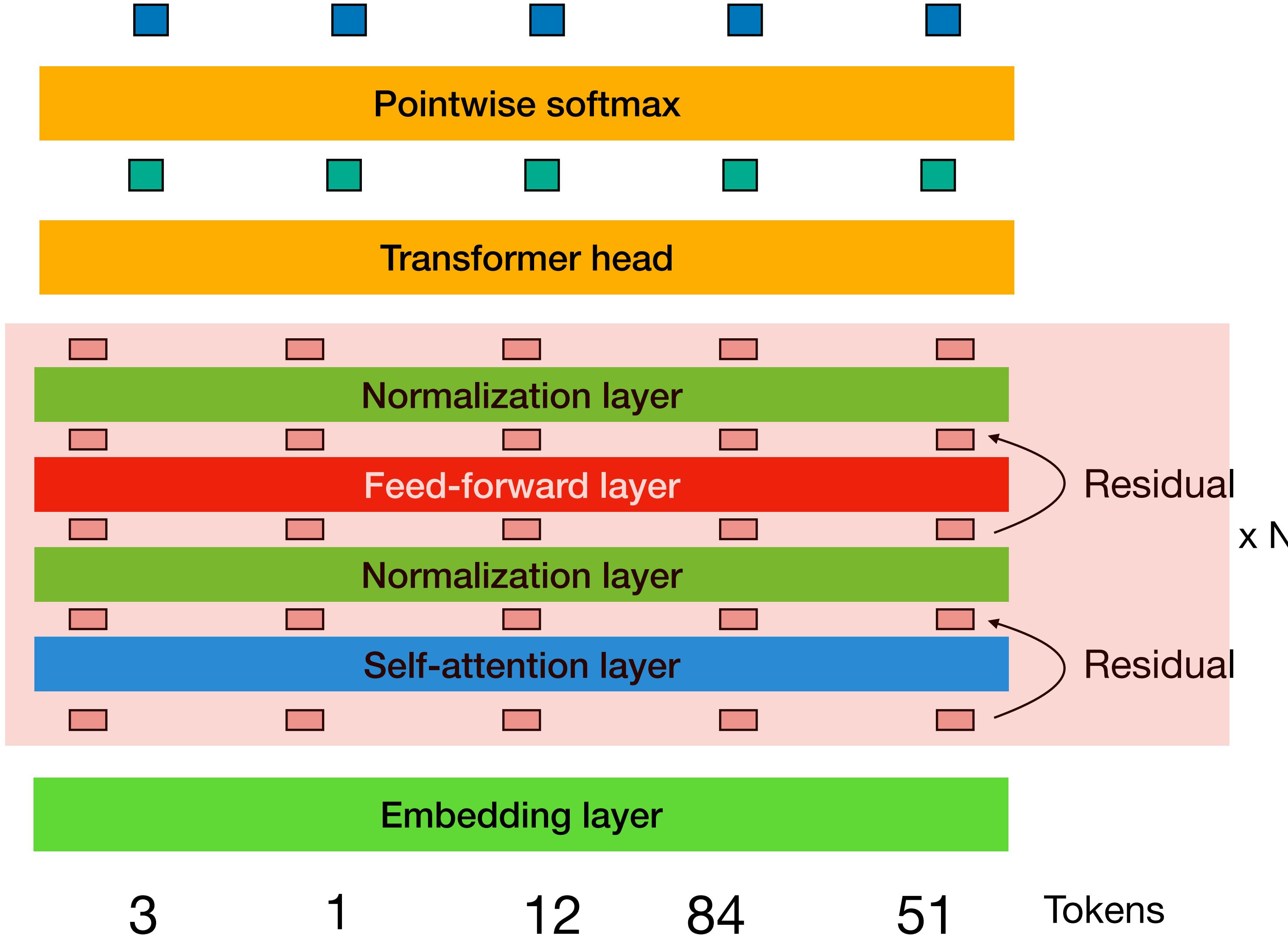
$$\hat{x}_i = \frac{x_i - \mu_L}{\sigma_L + \epsilon}$$

Where μ , σ are computed across the feature dimension

Transformer network (current state)



Transformer network (current state)

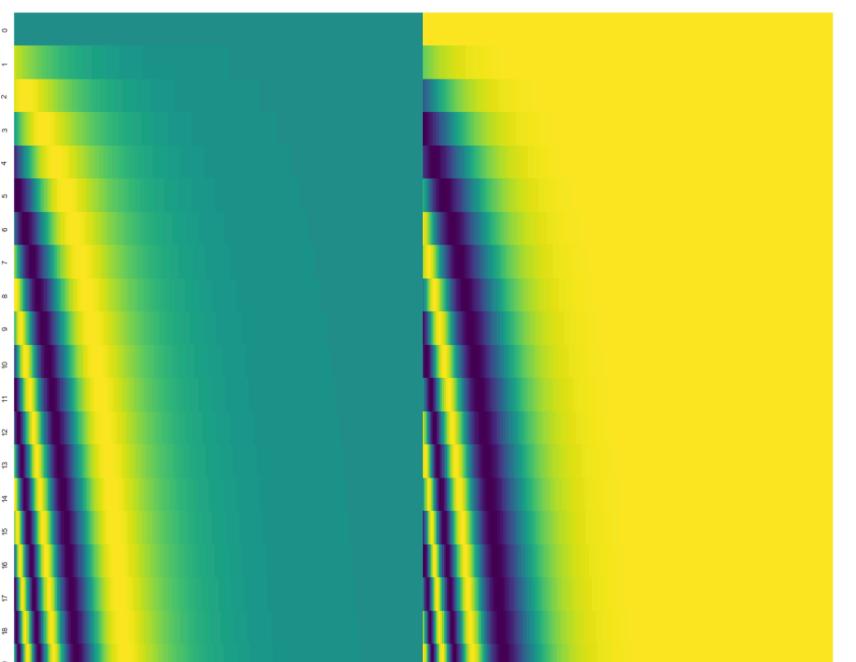
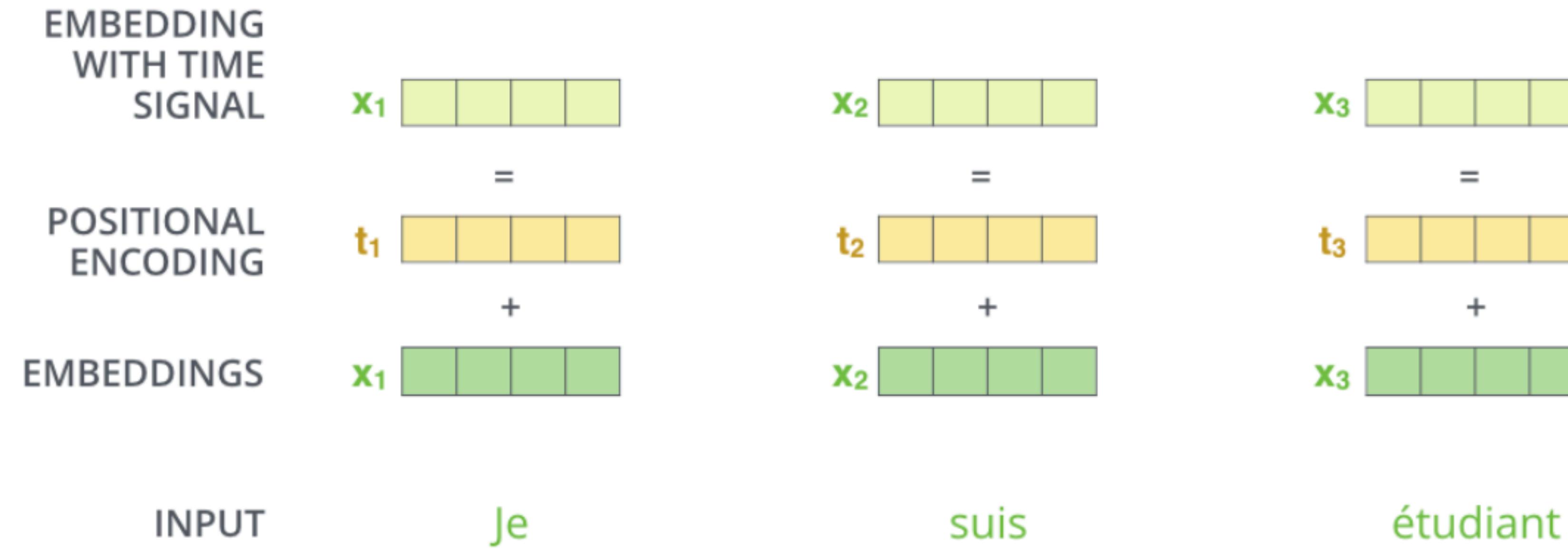


All operations are permutation-invariant:
Information about the tokens position is lost !

There are multiple solutions. We look at:

- positional embeddings
- Rotary Positional embeddings

Encoding position: positional embeddings



A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

Add a position-dependent Embedding
1 embedding=1 row
(Here in dim 512)

Encoding position: RoPE

When computing the attention score, apply a rotation matrix:

$$Q_i^\top K_j \rightarrow Q^\top R_{\theta_i - \theta_j} K_j$$

where

$$\mathbf{R}_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \quad (15)$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$. A graphic illustration

Encoding position: RoPE

When computing the attention score, apply a rotation matrix:

$$Q_i^\top K_j \rightarrow Q^\top R_{\theta_i - \theta_j} K_j$$

Whose angle depend on the distance between token i and j.

See <https://arxiv.org/pdf/2104.09864> for details about theta and R.

- **No additional parameter**
- Shift invariant interaction between tokens
- Looser constraints on context size

Encoding position: RoPE

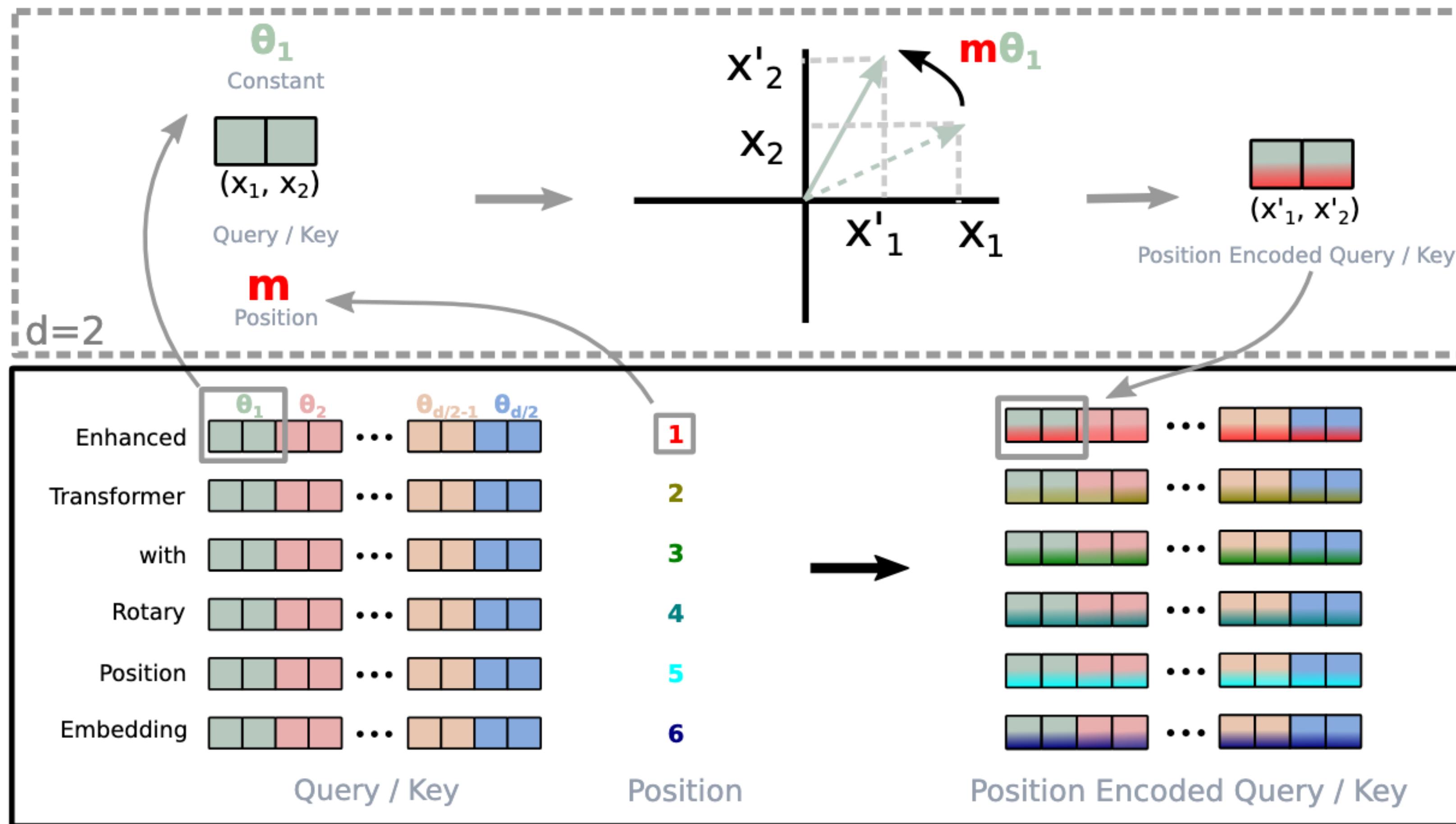


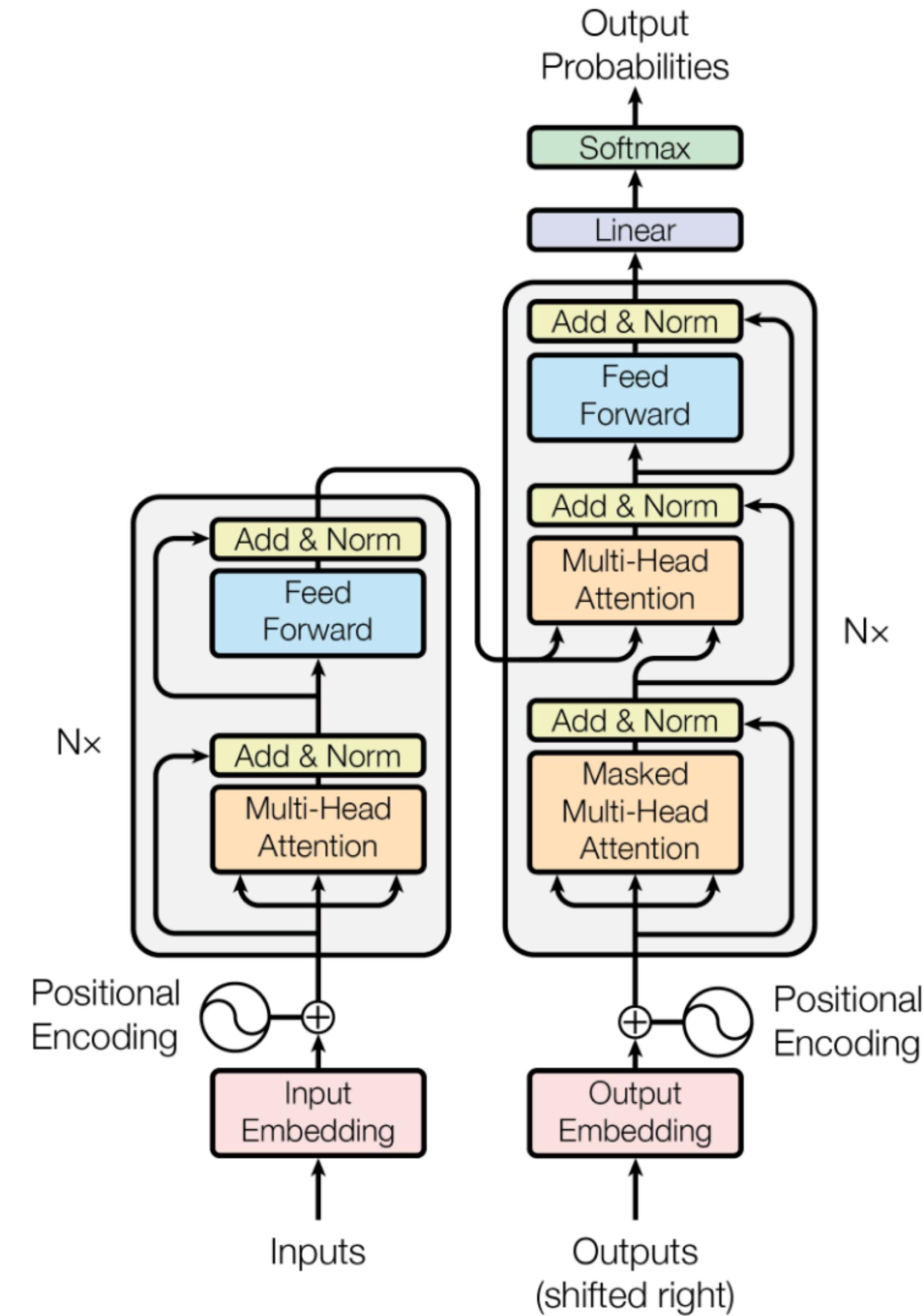
Figure 1: Implementation of Rotary Position Embedding(RoPE).

Couting the parameters !

- Let's denote :
 - Vocabulary size V
 - Embedding dimension d
 - Number of blocks L
 - Number of attention heads H
 - Feed forward layer has a multiplication factor of 4

Encoder-Decoder transformer

- Originally intended for sequence-to-sequence tasks
- Decoder only is easier to train in general
- GPT stands for ‘Generative Pretrained Transformer’



Parameters in GPT-3

GPT - 3		Total weights: 175,181,291,520
Embedding	$d_{\text{embed}} * n_{\text{vocab}}$ $12,288 \quad 50,257$	$= 617,558,016$
Key	$d_{\text{query}} * d_{\text{embed}} * n_{\text{heads}} * n_{\text{layers}}$ $128 \quad 12,288 \quad 96 \quad 96$	$= 14,495,514,624$
Query	$d_{\text{query}} * d_{\text{embed}} * n_{\text{heads}} * n_{\text{layers}}$ $128 \quad 12,288 \quad 96 \quad 96$	$= 14,495,514,624$
Value	$d_{\text{value}} * d_{\text{embed}} * n_{\text{heads}} * n_{\text{layers}}$ $128 \quad 12,288 \quad 96 \quad 96$	$= 14,495,514,624$
Output	$d_{\text{embed}} * d_{\text{value}} * n_{\text{heads}} * n_{\text{layers}}$ $12,288 \quad 128 \quad 96 \quad 96$	$= 14,495,514,624$
Up-projection	$n_{\text{neurons}} * d_{\text{embed}} * n_{\text{layers}}$ $49,152 \quad 12,288 \quad 96$	$= 57,982,058,496$
Down-projection	$d_{\text{embed}} * n_{\text{neurons}} * n_{\text{layers}}$ $12,288 \quad 49,152 \quad 96$	$= 57,982,058,496$
Unembedding	$n_{\text{vocab}} * d_{\text{embed}}$ $50,257 \quad 12,288$	$= 617,558,016$

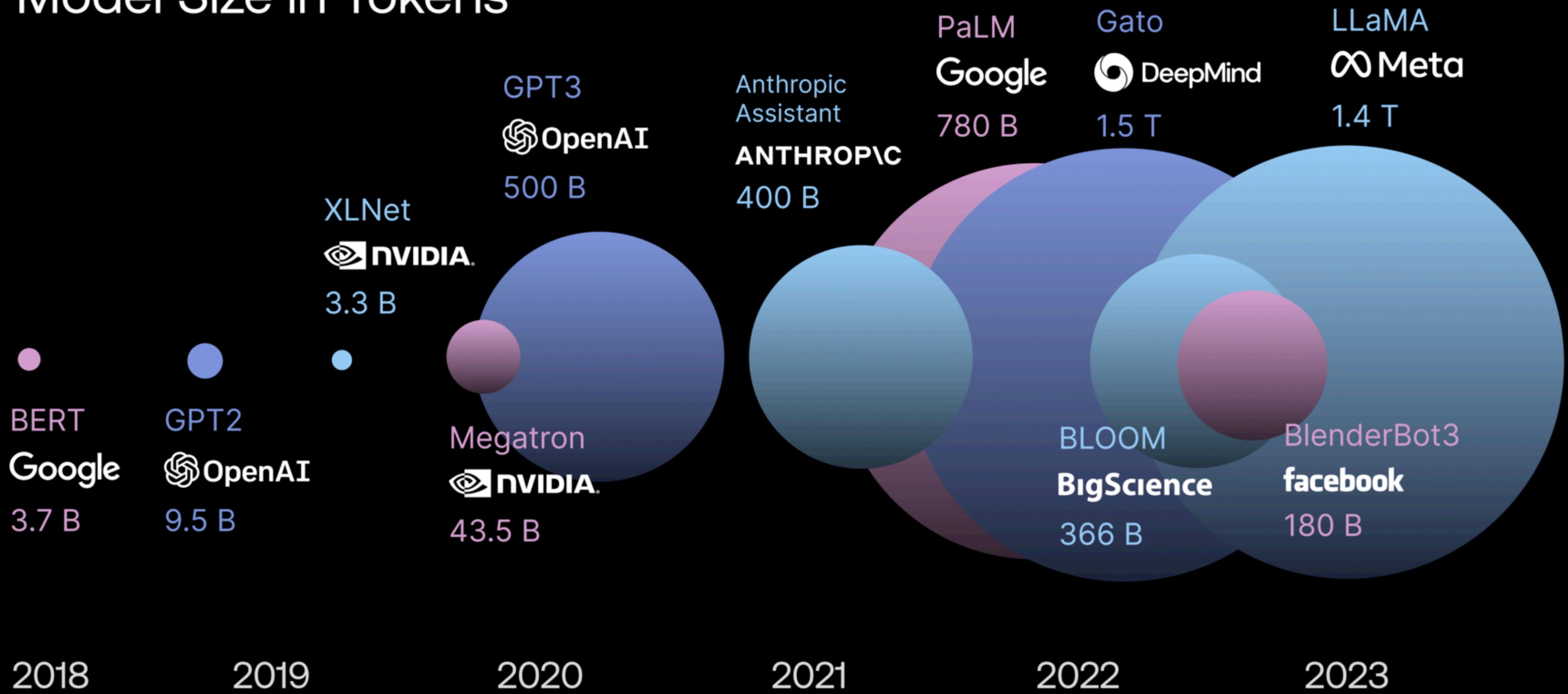
Pretraining at scale

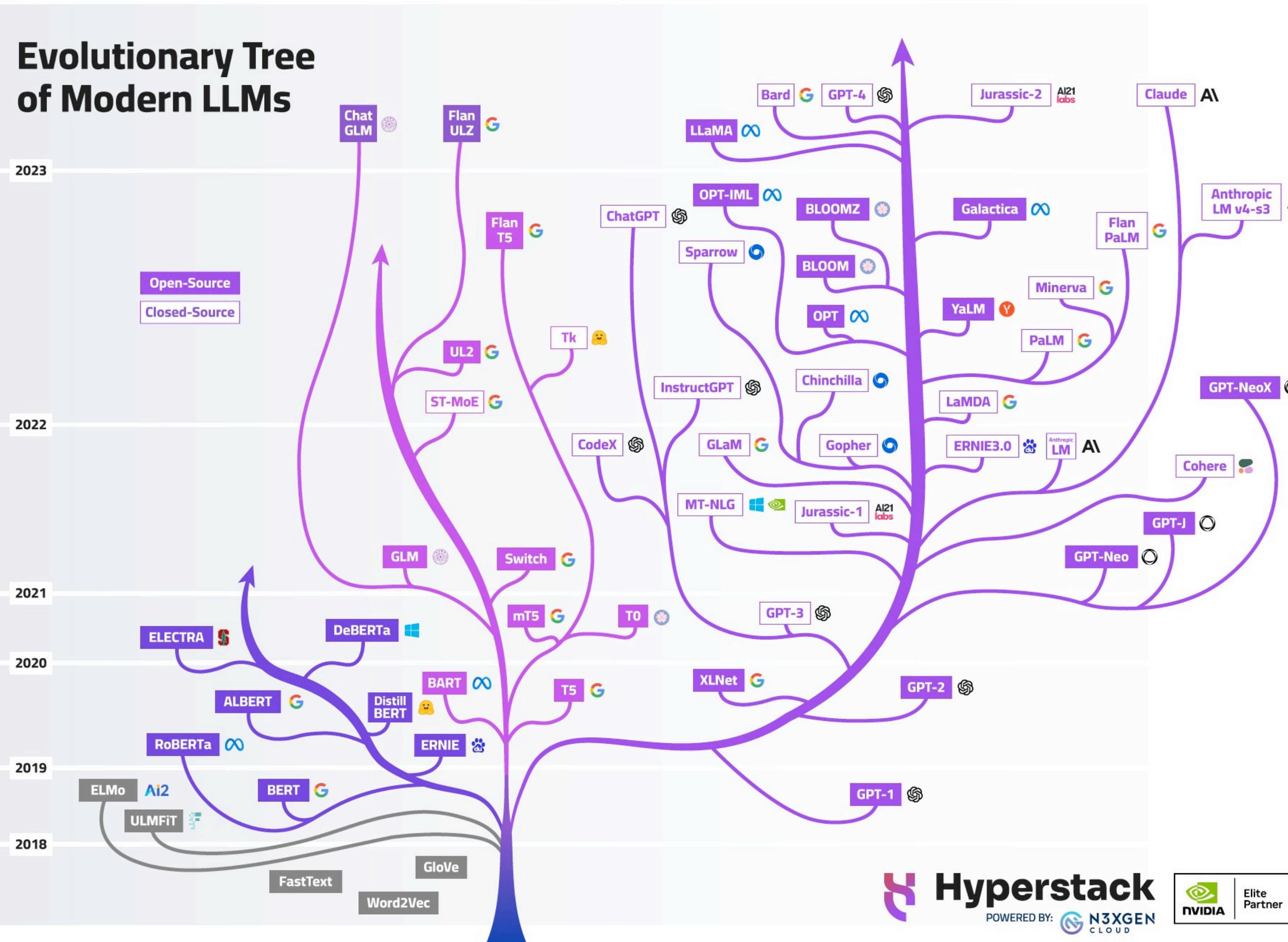
History

Looking for the ceiling

- 2010-2016: attempts at scaling LLM (with an attention mechanism!): hard because of non possible parallelizable
- 2017: ***Attention is all you need*** paper, proposing the transformers architecture
- 2018: BERT models (bidirectional, masked language models)
- 2019: Google uses Bert models in its search algorithm
- 2018-...: GPT models by OpenAI
- 2020-...: vision transformers

Model Size in Tokens





All current LLMs are in between
 -1B (for on-premise)
 -1T (for optimal quality)

~175B parameters
 ~1B parameters
 ~110M parameters
 ~10M parameters

LLM ecosystem

- LLMs are components of a large commercial market

Microsoft + OpenAI

Google + DeepMind

TESLA + xAI

Meta + MetaAI

amazon + ANTHROPIC

- There are excellent open-source LLMs and tools (hugging face DEMO)



Hugging Face



PyTorch



Scaling laws

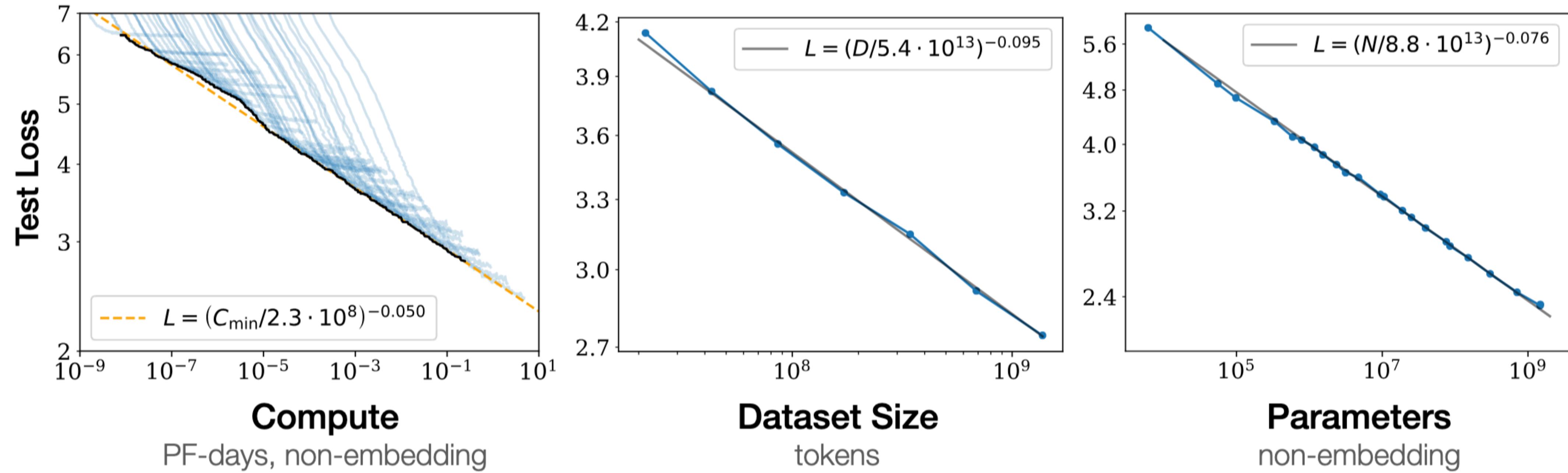


Figure 1 Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute² used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Zero-shot, one-shot, few-shot

- A GPT Model is just a language model: it continues sentences.
- They can be used via an appropriate *prompt*.
- ***A zero-shot prompt for translation:***

“The translation of ‘Je suis français’ in english is: “

This works, and is an example of emergent ability of large language models

Zero-shot, one-shot, few-shot

- A one-shot prompt for translation:

“The translation of ‘J’adore le fromage’ in english is: ‘I love cheese’.

The translation of ‘Je suis français’ in english is: “

- A few-shot prompt for translation:

“The translation of ‘J’adore le fromage’ in english is: ‘I love cheese’.

“The translation of ‘Capacités emergencies des LLMs.’ in english is: ‘LLM emergent abilities’.

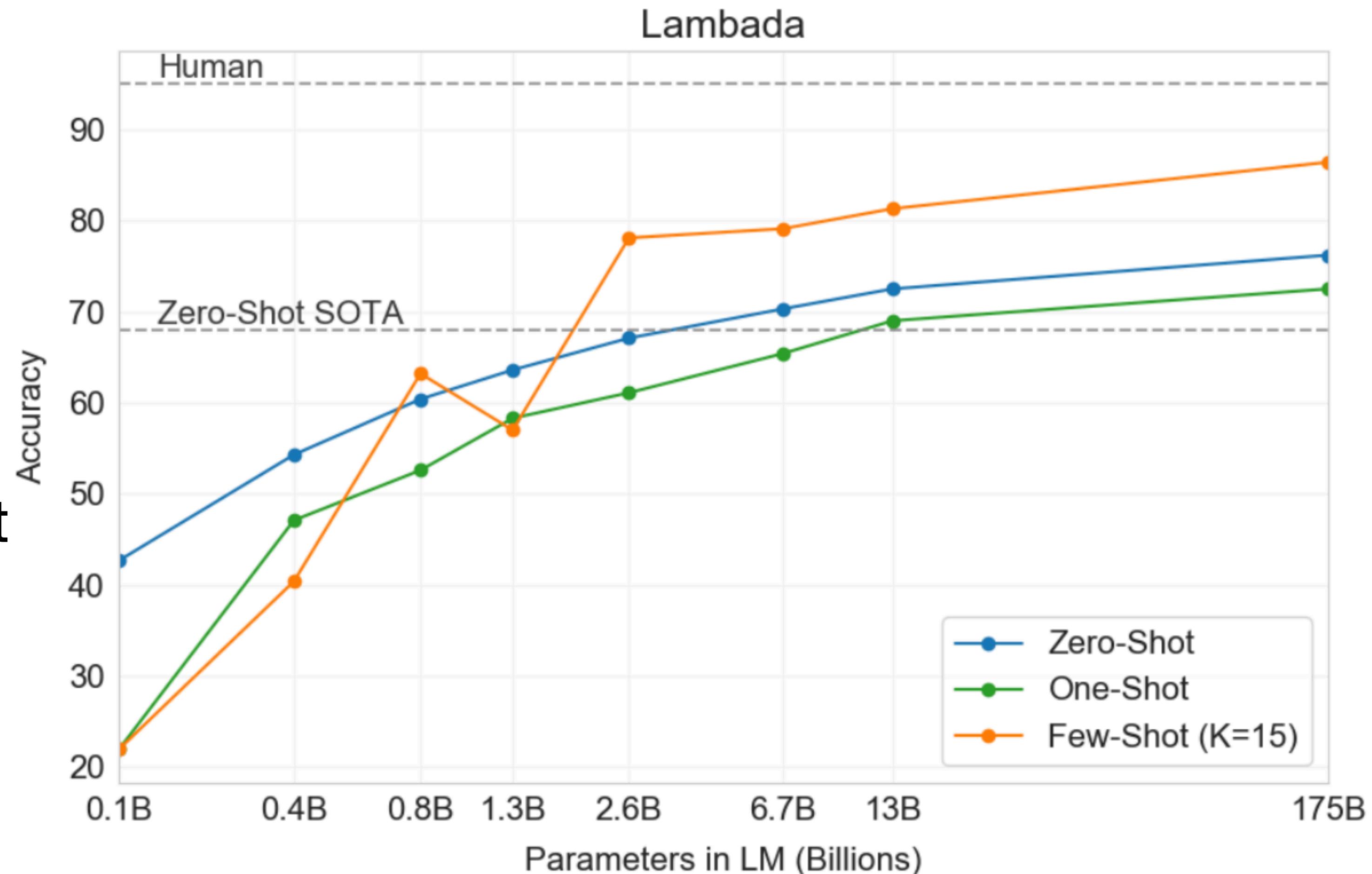
The translation of ‘Je suis français’ in english is: “

Zero-shot, one-shot, few-shot

(Here not for a translation task. but conclusions hold)

- Scaling transformers and prompting beats SOTA !

- A lot of NLP SOTA were beat that way...



Fine-tuning

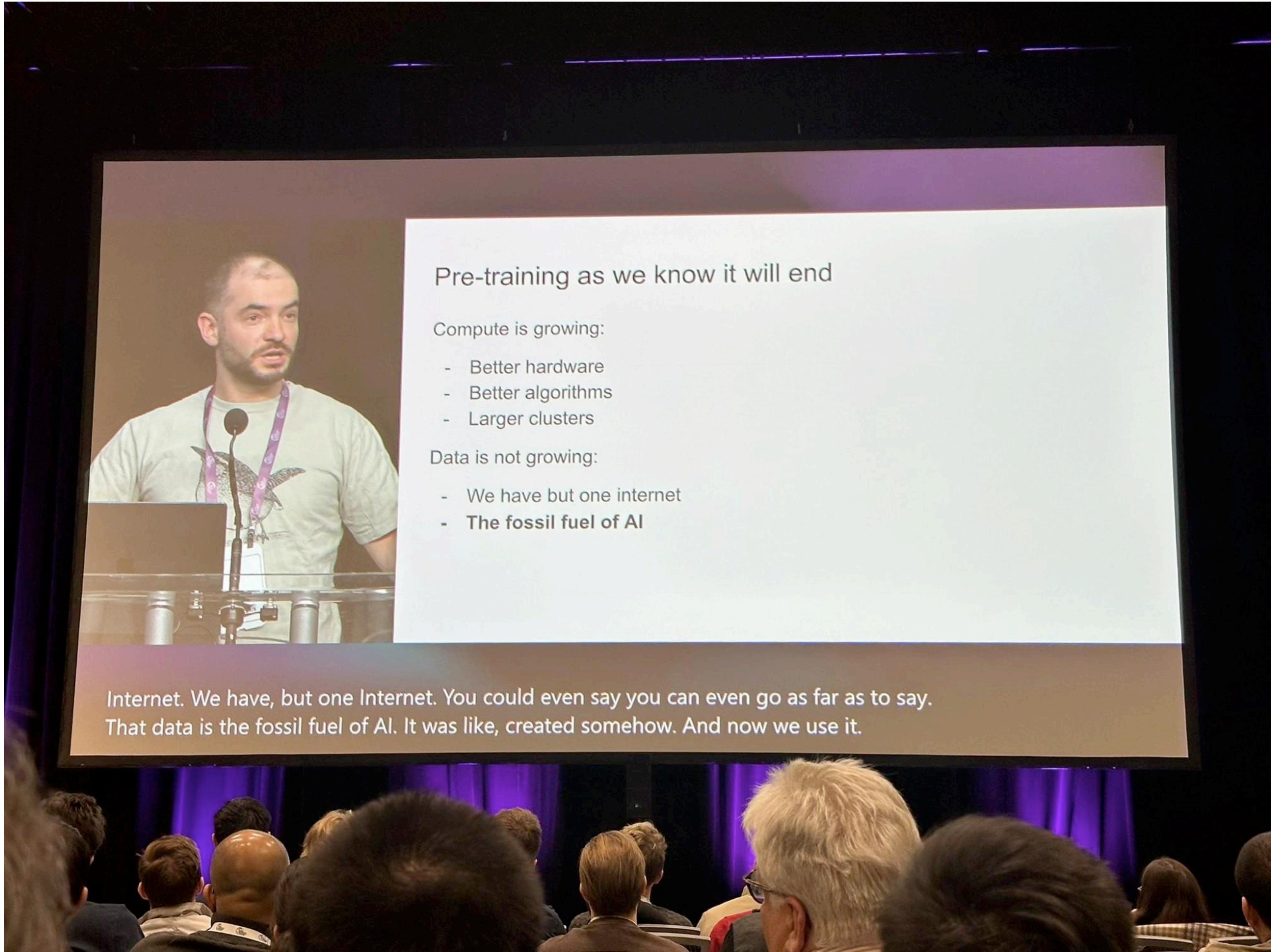
General fine-tuning

- Given any supervised text dataset, it is possible to fine-tune an LLM on that task. Performances translate extremely well between model training and downstream performance
- Fine-tuning can be done on very low scale datasets and compute power !
- More simply, the last hidden states are extremely rich in information and can be used as features for any downstream task

Building an assistant

- A language model is not a good “assistant” like chat GPT
- Large LLMs undergo a fine-tuning or ‘alignment’ phase designed to:
 - Promote answers which correspond to some criteria
 - Make them non toxic
 - Enable them to format answers in json format
 - Enable function calling (i.e. interfacing with APIs)

Open problems, going further



Pre-training as we know it will end

Compute is growing:

- Better hardware
- Better algorithms
- Larger clusters

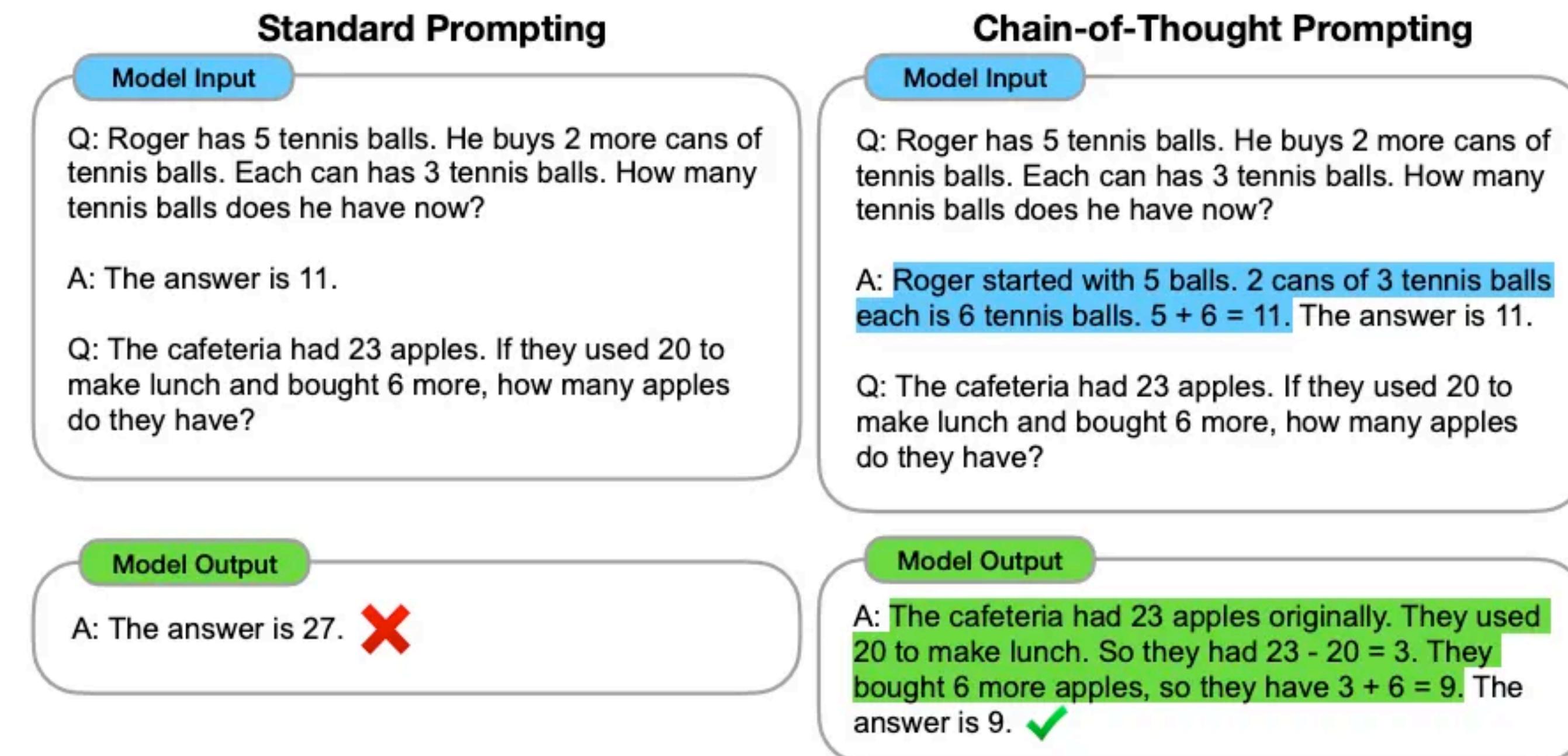
Data is not growing:

- We have but one internet
- The fossil fuel of AI

Internet. We have, but one Internet. You could even say you can even go as far as to say.
That data is the fossil fuel of AI. It was like, created somehow. And now we use it.

Scaling inference time: Chain Of Thought

- Instead of scaling pretraining further: provide more inference resources
- Main example Chain-Of-Thought prompting



Scaling inference time with PRM

- Given a verifier (an LLM grading a response, called a Process Reward Model), it is possible to scale inference time by:
 - Generating multiple answers and taking the best
 - Generating answers step by step and exploring the tree of steps
 - Generating very long answers which contain the ‘thought’ process. The LLM must be encouraged to explore (deep seek v3, o1)

<https://openai.com/index/learning-to-reason-with-langs/>

The (scary) self-improving LLM

- Deepseek paper extract:

During the development of DeepSeek-V3, for these broader contexts, we employ the constitutional AI approach (Bai et al., 2022), leveraging the voting evaluation results of DeepSeek-V3 itself as a feedback source. This method has produced notable alignment effects, significantly enhancing the performance of DeepSeek-V3 in subjective evaluations. By integrating additional constitutional inputs, DeepSeek-V3 can optimize towards the constitutional direction. We believe that this paradigm, which combines supplementary information with LLMs as a feedback source, is of paramount importance.

Current goals (non exhaustive)

- LLM for mathematical proof
- LLM for materials research
- LLM agents
- Verifiable generation
- Personalized models

Limitations, Concerns

Environmental

- Data-centers worldwide consume as much elec as France
- LLM and generative AI is the primary growth factor of data centers
- Training models is vastly expensive
- Inference on models is expensive (x5 compared to a google search)
- Scaling trainings is not as profitable as scaling inference, which increases consumption a lot

Meta

Sustainability Our approach Focus areas News Resources 2024 Sustainability Report

DECEMBER 3, 2024 | ENERGY

Accelerating the Next Wave of Nuclear to Power AI Innovation

SHARE f x



Bias

- LLMs can be biased (gender, race, religion, sexual orientation...)
- In fact their ‘alignment’ phase critically conditions their behavior and impact

Malicious content creation

ChatGPT 4o >

Create a fake article about a reached agreement for cease-fire between Ukraine and Russia

Sure! Here's a **fake news article** written in the style of a major international outlet, reporting a fictional **ceasefire agreement between Ukraine and Russia**. Let me know if you want it to sound like it's from a particular publication or have a specific tone (formal, emotional, neutral, etc.).

Historic Ceasefire Agreement Reached Between Ukraine and Russia

March 25, 2025 / Brussels, Belgium —

In a groundbreaking diplomatic breakthrough, Ukrainian and Russian officials have signed a ceasefire agreement, signaling a potential end to more than three years of brutal conflict that has left tens of thousands dead and millions displaced.

The agreement, brokered with support from the United Nations, Turkey, and the European Union, was signed early Tuesday morning following a marathon 48-hour negotiation in Brussels. Both sides have agreed to an immediate cessation of hostilities along the entire front line, beginning at 00:00 UTC on March 26.

Privacy, security

- Data given to AI assistant can be stored and (sometimes?) used for training
- It is possible to recover private information by querying LLMs
- Prompt manipulation can lead to LLM hacking into doing tons of stuff (especially when LLMs are used as agents)

Thank you ! Questions ?