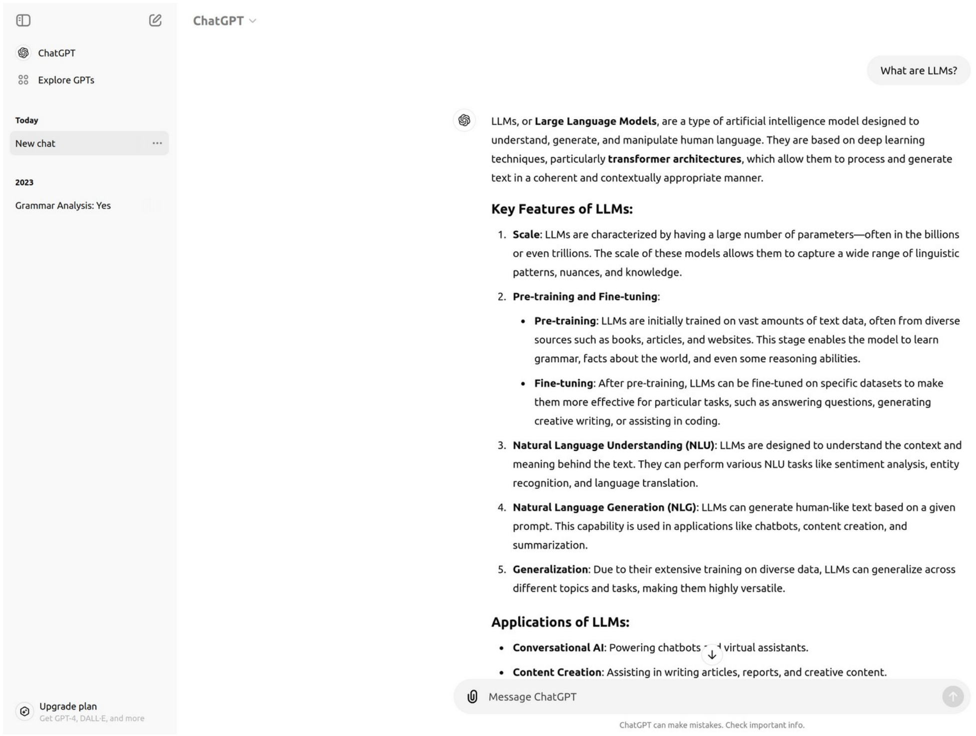
Large Language Models for Linguists: A Practical Introduction

Tommi Buder-Gröndahl

Large Language Models (LLMs)

- Large *deep neural networks* (DNNs), currently mostly *Transformers* (Vaswani et al. 2017)
- BERT, GPT, Llama, ...
- Pre-trained on generic linguistic tasks: e.g. predicting masked words or upcoming text



Large Language Models (LLMs)

- Claimed to attain linguistic competence without innate language-specific capacities
- Vs. rule-based NLP, generative linguistics

Emergent linguistic structure in artificial neural networks trained by self-supervision

Christopher D. Manning[a,1], Kevin Clark[a], John Hewitt[a], Urvashi Khandelwal[a], and Omer Levy[b]

Finding Universal Grammatical Relations in Multilingual BERT

Ethan A. Chi, John Hewitt, and **Christopher D. Manning**
Department of Computer Science
Stanford University
{ethanchi,johnhew,manning}@cs.stanford.edu

Large Language Models Demonstrate the Potential of Statistical Learning in Language

Pablo Contreras Kallens, Ross Deans Kristensen-McLachlan, Morten H. Christiansen

First published: 25 February 2023 | https://doi.org/10.1111/cogs.13256 | Citations: 2

This article is part of the "Progress & Puzzles of Cognitive Science" letter series.

Modern language models refute Chomsky's approach to language

Steven T. Piantadosi[a,b]
[a]UC Berkeley, Psychology [b]Helen Wills Neuroscience Institute

Tutorial schedule

5.9. Theoretical & mathematical background

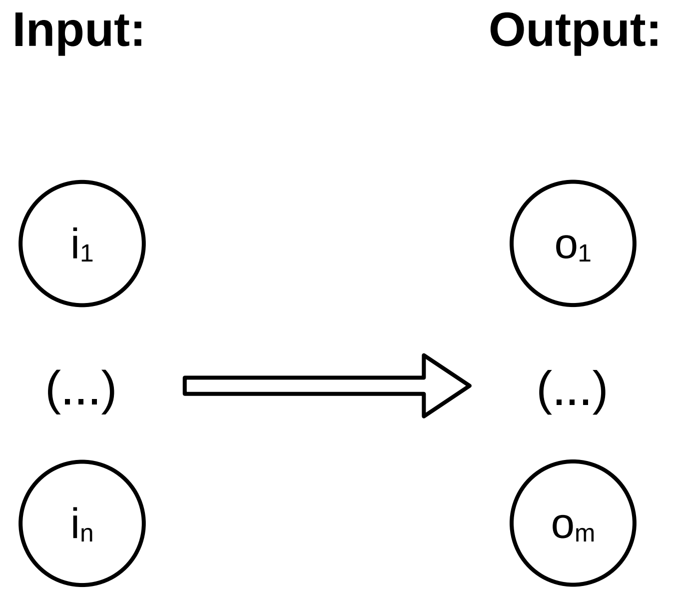
6.9. Practical session (Jupyter notebook)

Today's lecture structure

1. Neural networks: Basics
2. Recurrent neural networks (RNNs)
3. Attention
4. Transformer
5. Large Language Models (LLMs)
6. Interpreting LLMs

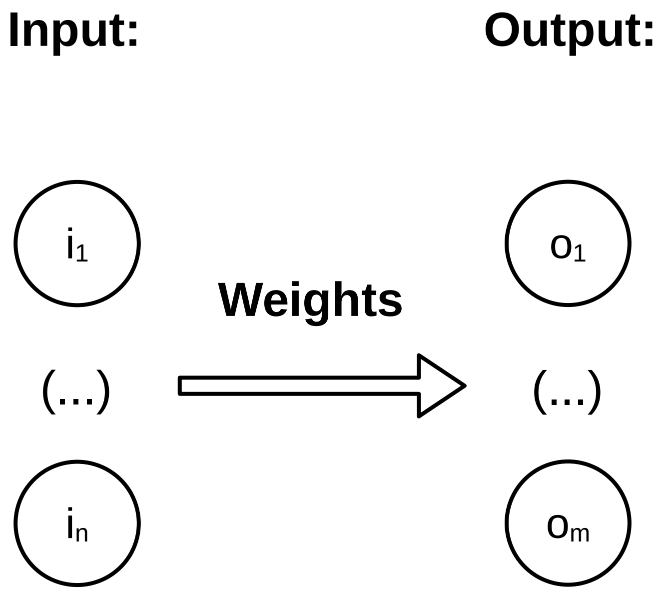
Neural networks: Basics

Neural network



Neural network

Input vector:
i = [i_1, …, i_n]



Output vector:
o = [o_1, …, o_m]

Mathematical concepts

Vector: sequence of numbers

$$v = [v_1, \ldots, v_n]$$
$$w = [w_1, \ldots, w_n]$$

Mathematical concepts

Vector: sequence of numbers

$$v = [v_1, \ldots, v_n]$$
$$w = [w_1, \ldots, w_n]$$

Vector operations:

- Sum: $v + w = [v_1+w_1, \ldots, v_n+w_n]$
- Multiplication with a scalar: $av = [av_1, \ldots, av_n]$
- Dot product: $v \cdot w = v_1 w_1 + \ldots + v_n w_n$

Mathematical concepts

Matrix: table of numbers with rows and columns

$$A = \underbrace{\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}}_{n \text{ columns}} \left.\vphantom{\begin{bmatrix} a_{11} \\ \dots \\ a_{m1} \end{bmatrix}}\right\} m \text{ rows}$$

Mathematical concepts

Matrix: table of numbers with rows and columns

$$A=\underbrace{\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}}_{n \text{ columns}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \end{bmatrix}}\right\} m \text{ rows} \qquad B=\underbrace{\begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}}_{k \text{ columns}} \left.\vphantom{\begin{bmatrix} b \\ b \\ b \end{bmatrix}}\right\} n \text{ rows}$$

Matrix multiplication:

$$AB=\underbrace{\begin{bmatrix} a_{11}b_{11}+\dots+a_{1n}b_{n1} & \dots & a_{11}b_{1k}+\dots+a_{1n}b_{nk} \\ \dots & \dots & \dots \\ a_{m1}b_{11}+\dots+a_{mn}b_{n1} & \dots & a_{m1}b_{1k}+\dots+a_{mn}b_{nk} \end{bmatrix}}_{k \text{ columns}} \left.\vphantom{\begin{bmatrix} a \\ a \\ a \end{bmatrix}}\right\} m \text{ rows}$$

Mathematical concepts

Matrix: table of numbers with rows and columns

$$A=\underbrace{\left.\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}\right\}}_{n \text{ columns}} m \text{ rows}$$

$$B=\underbrace{\left.\begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}\right\}}_{k \text{ columns}} n \text{ rows}$$

Transposition:

$$A^T=\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nm} \end{bmatrix}$$

$$A^T[i,j]=A[j,i]$$

Matrix multiplication:

$$AB=\underbrace{\left.\begin{bmatrix} a_{11}b_{11}+\dots+a_{1n}b_{n1} & \dots & a_{11}b_{1k}+\dots+a_{1n}b_{nk} \\ \dots & \dots & \dots \\ a_{m1}b_{11}+\dots+a_{mn}b_{n1} & \dots & a_{m1}b_{1k}+\dots+a_{mn}b_{nk} \end{bmatrix}\right\}}_{k \text{ columns}} m \text{ rows}$$

Neural network

Input vector:

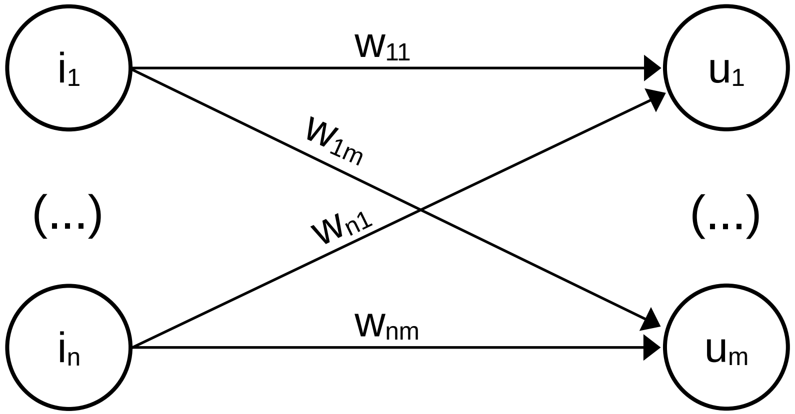
$\mathbf{i} = [i_1, \ldots, i_n]$

Weight matrix:

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \cdots & \cdots & \cdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix}$$

Weighted sums of inputs:

$\mathbf{u} = \ \mathbf{i}W = [u_1, \ldots, u_m]$



Neural network

Input vector:

$\mathbf{i} = [i_1, \ldots, i_n]$

Weight matrix:

$$W = \begin{bmatrix} w_{11} & \ldots & w_{1m} \\ \ldots & \ldots & \ldots \\ w_{n1} & \ldots & w_{nm} \end{bmatrix}$$

Weighted sums of inputs:

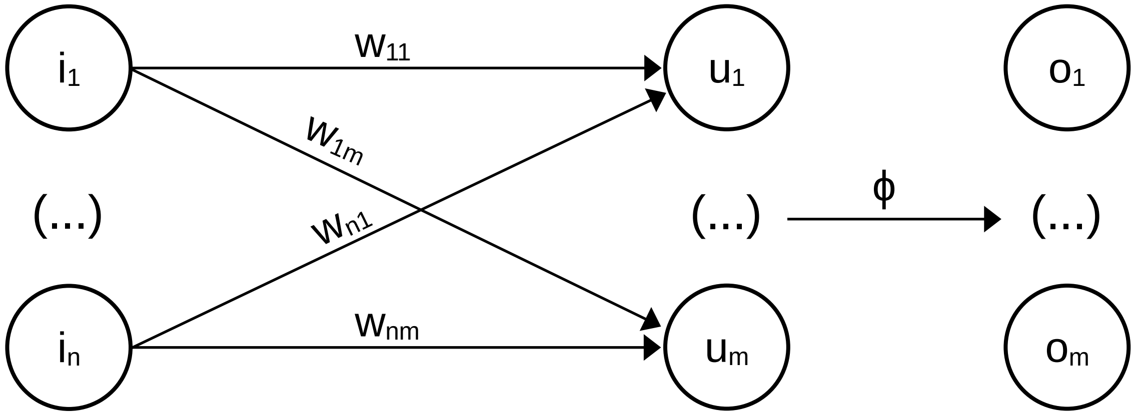
$\mathbf{u} = \mathbf{i}W = [u_1, \ldots, u_m]$

Activation function:

$\phi \in \{\sigma, \tanh, \mathrm{ReLu}, \ldots\}$

Output:

$\mathbf{o} = \phi(\mathbf{u}) = [o_1, \ldots, o_m]$



Neural network

Input vector:

$\mathbf{i} = [i_1, \ldots, i_n]$

Weight matrix:

$$W = \begin{bmatrix} w_{11} & \ldots & w_{1m} \\ \ldots & \ldots & \ldots \\ w_{n1} & \ldots & w_{nm} \end{bmatrix}$$

Weighted sums of inputs:

$\mathbf{u} = \mathbf{i}W = [u_1, \ldots, u_m]$

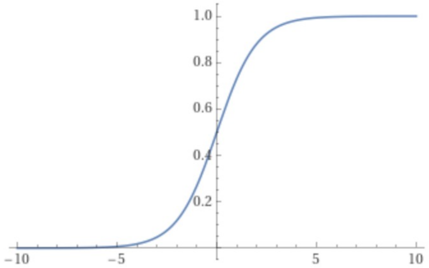
Activation function:

$\phi \in \{\sigma, \tanh, \mathrm{ReLu}, \ldots\}$

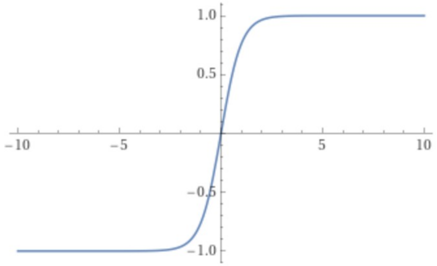
Output:

$\mathbf{o} = \phi(\mathbf{u}) = [o_1, \ldots, o_m]$

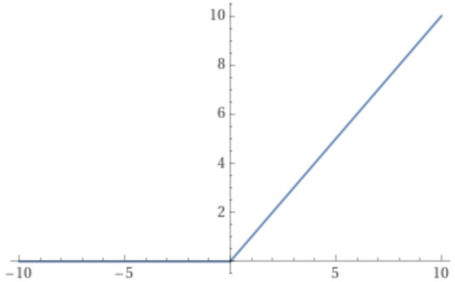
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



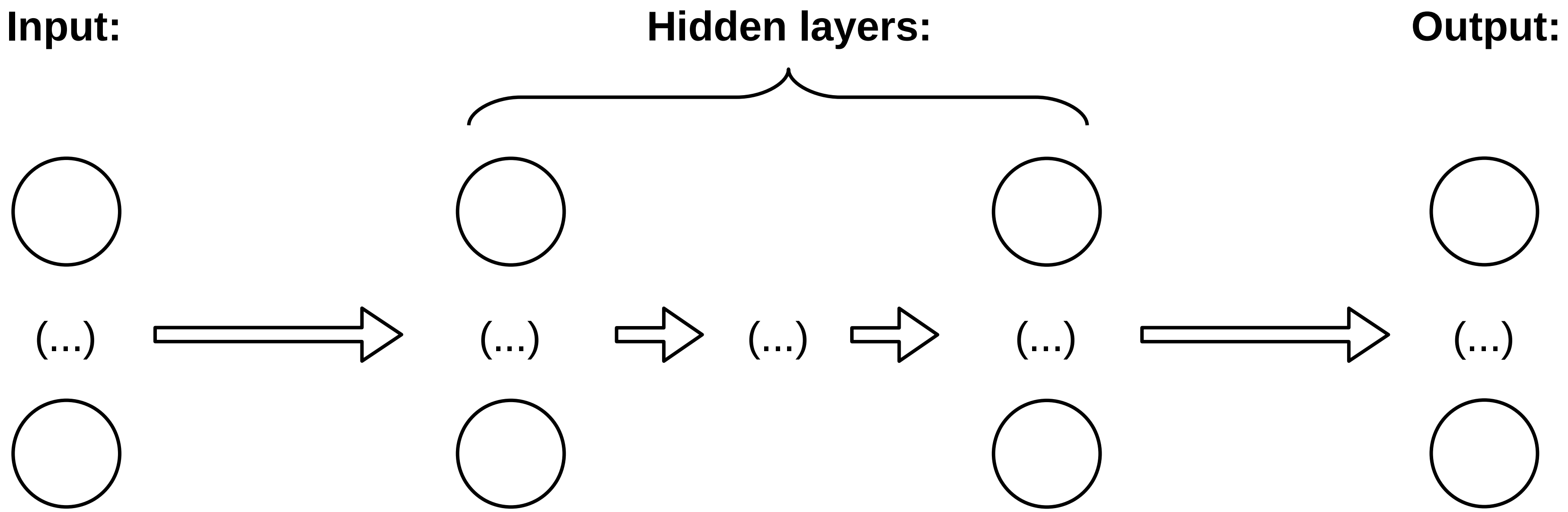
$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



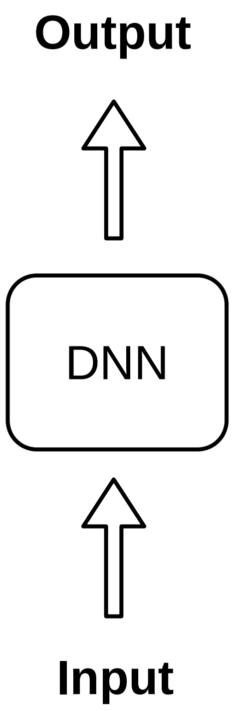
$$ReLU(x) = max(0, x)$$



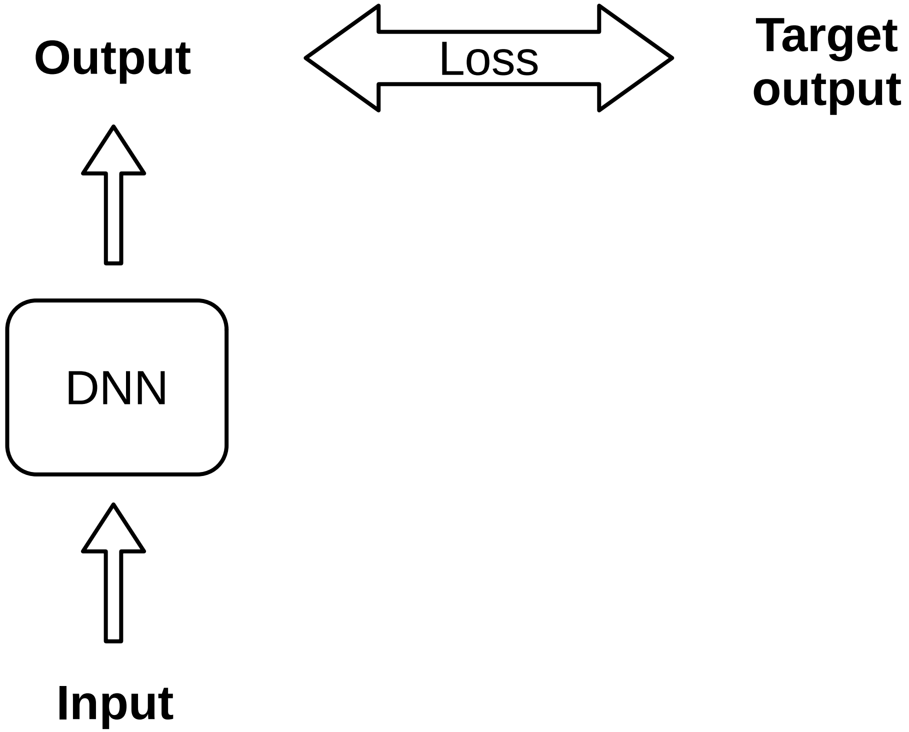
Deep Neural Network (DNN)



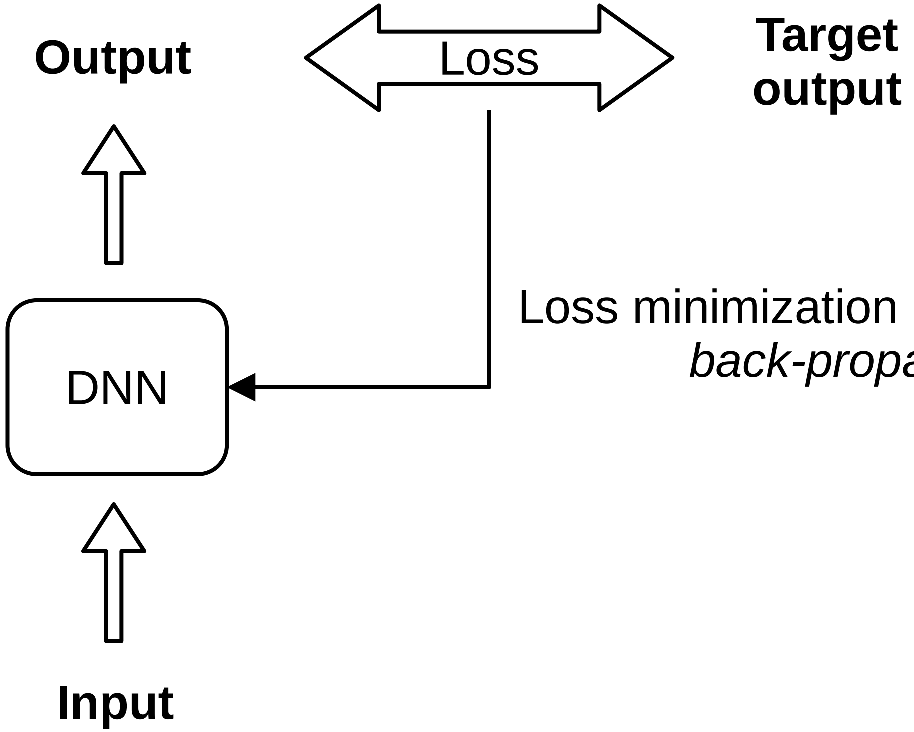
Training a DNN



Training a DNN

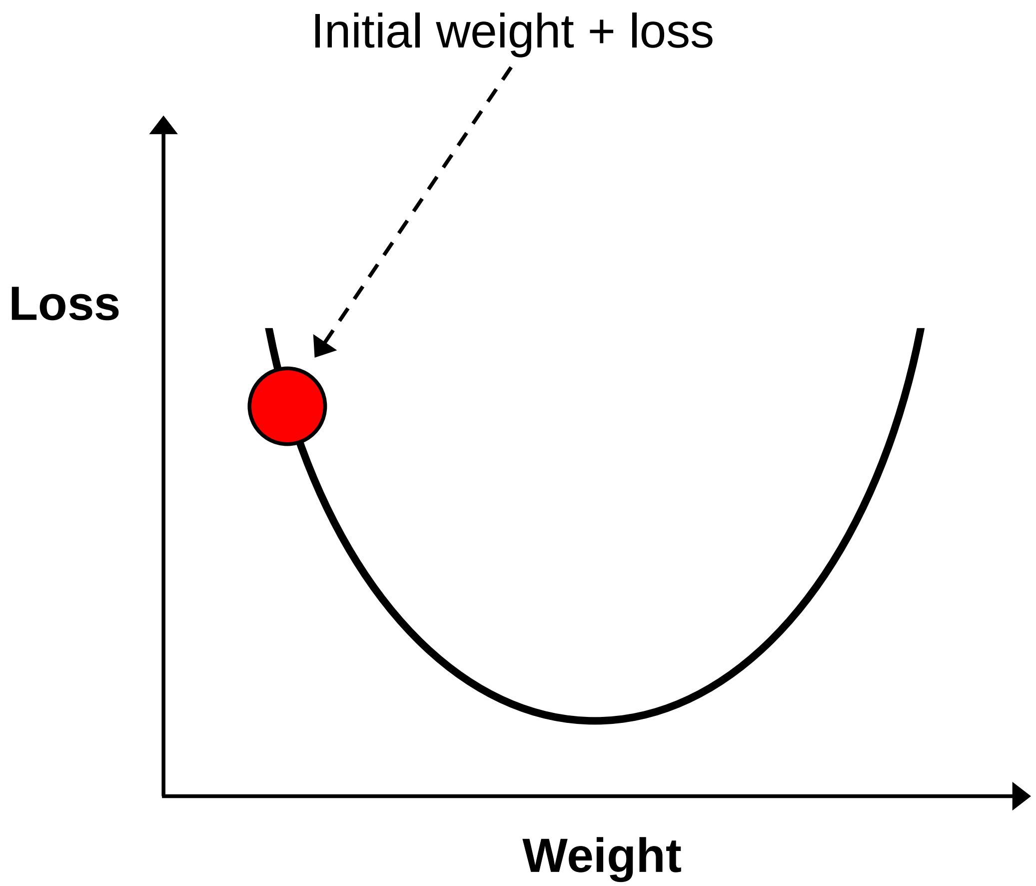


Training a DNN

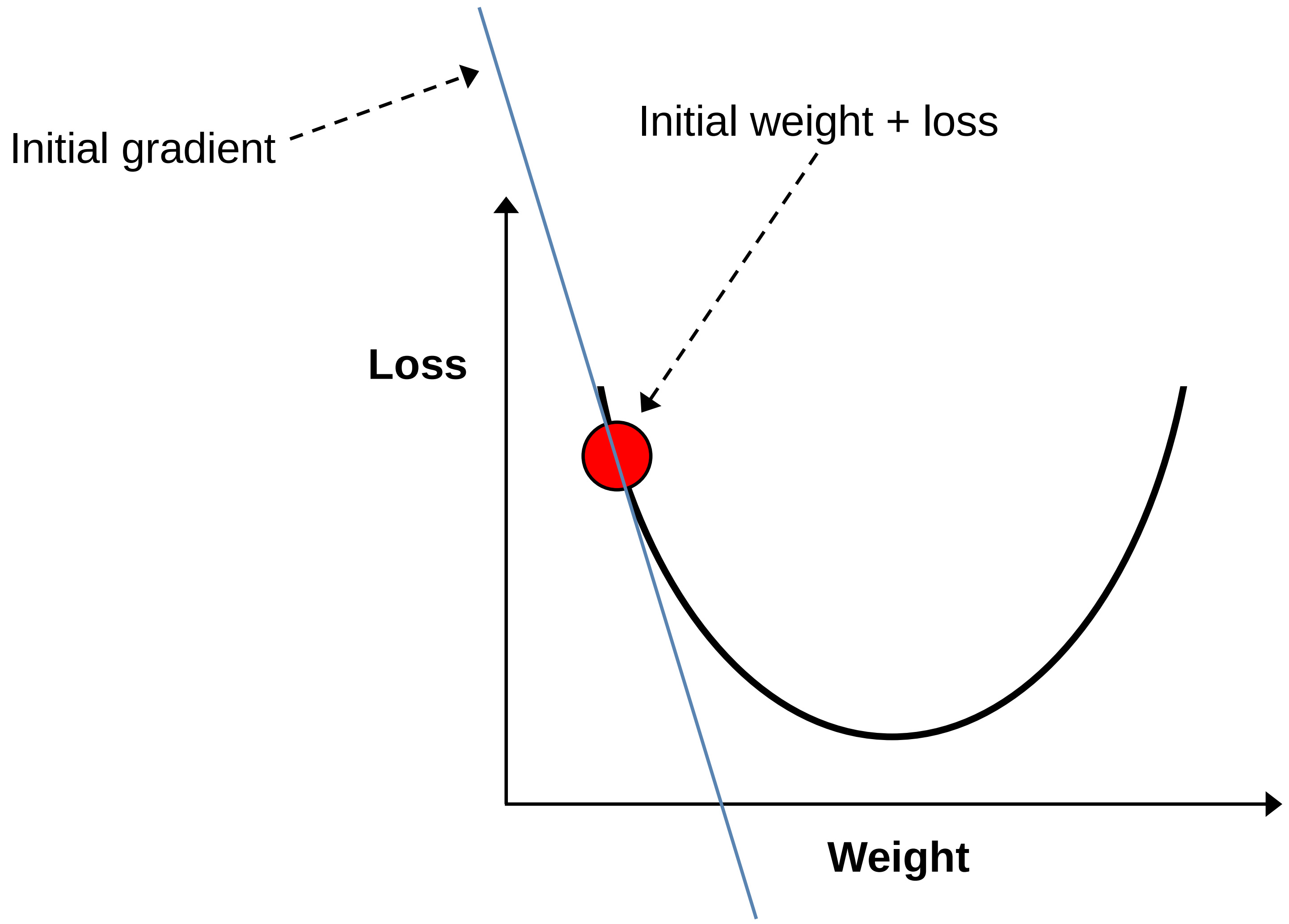


Loss minimization by modifying weights =
back-propagation of error

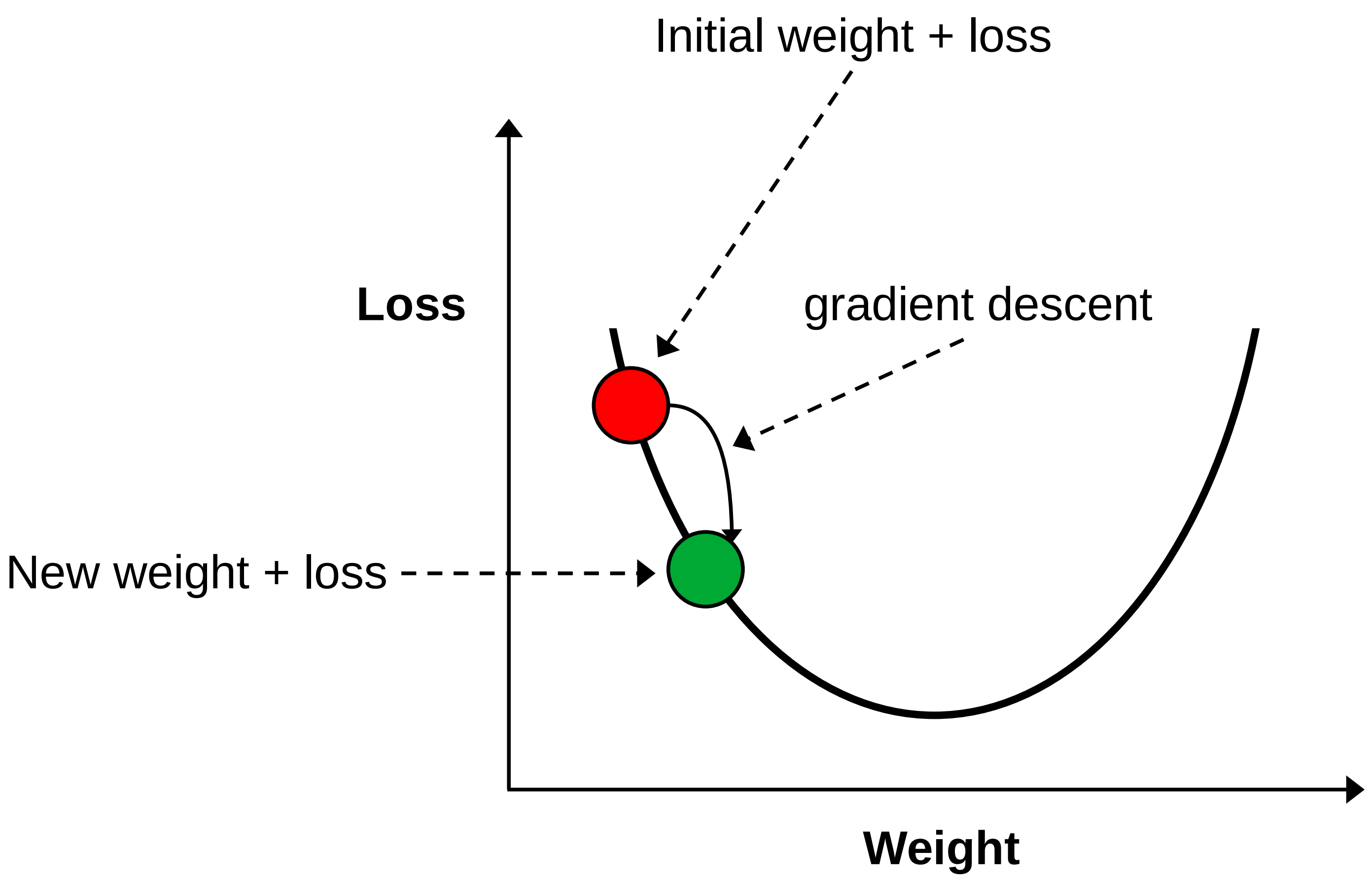
Gradient descent



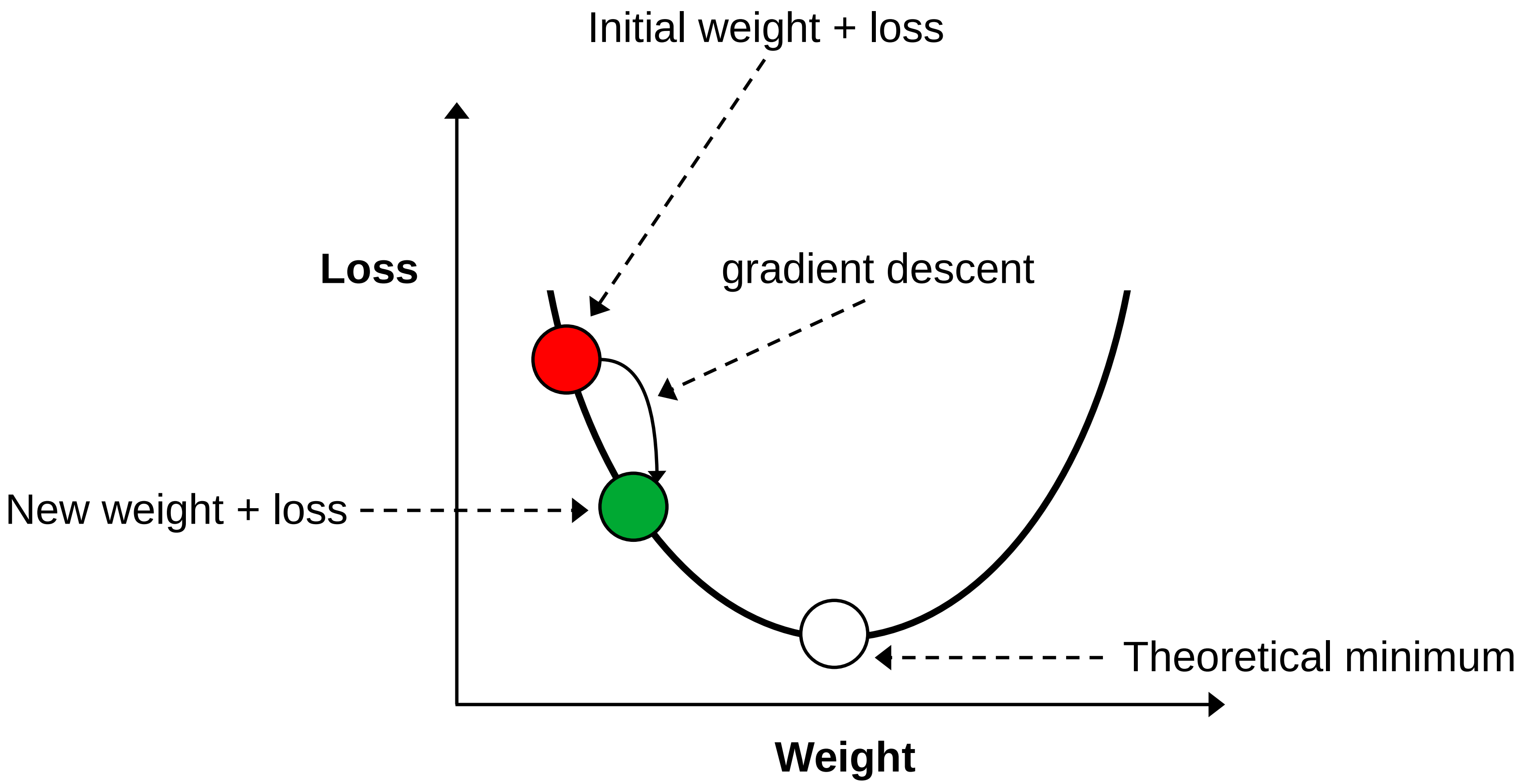
Gradient descent



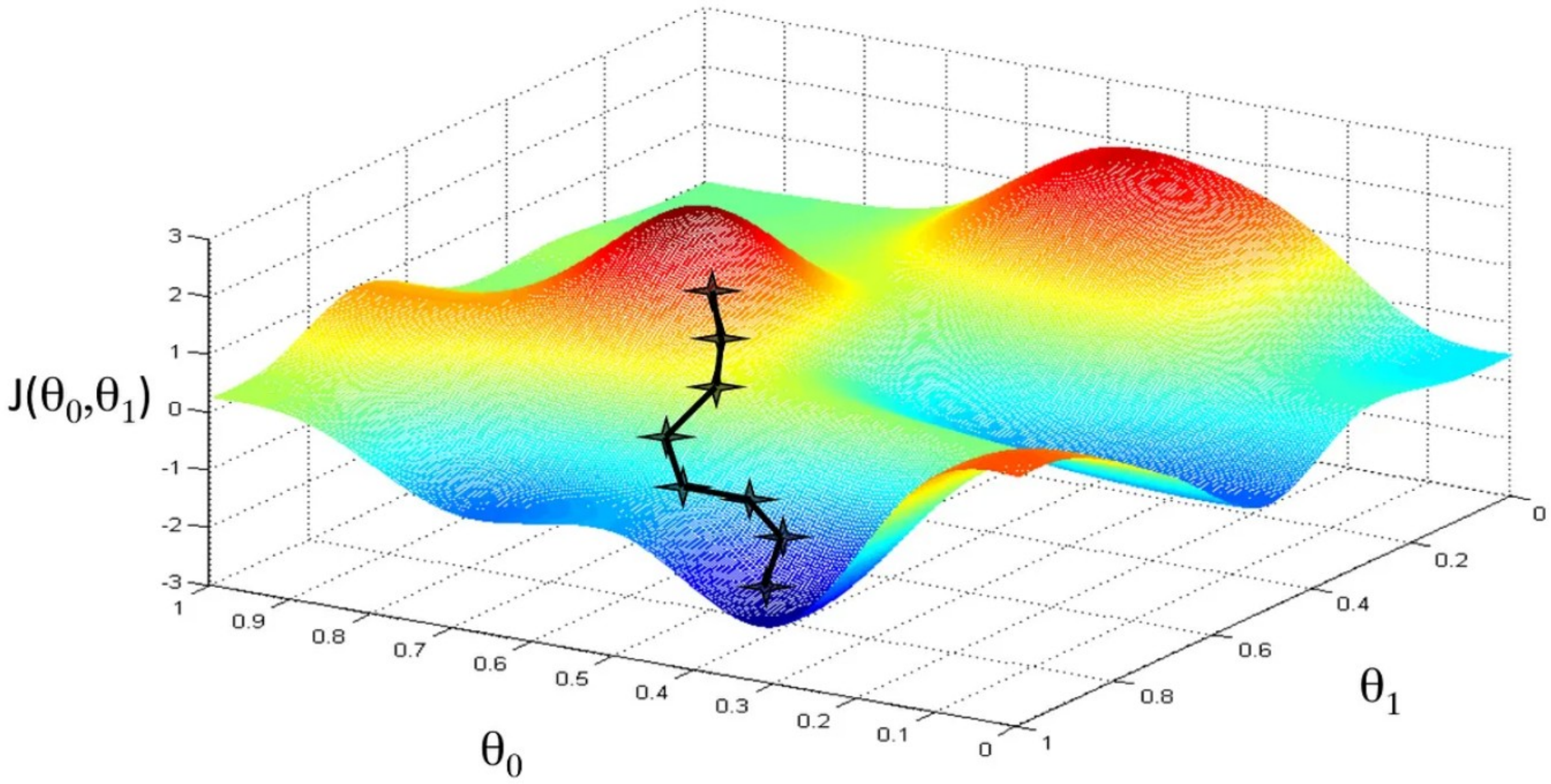
Gradient descent



Gradient descent

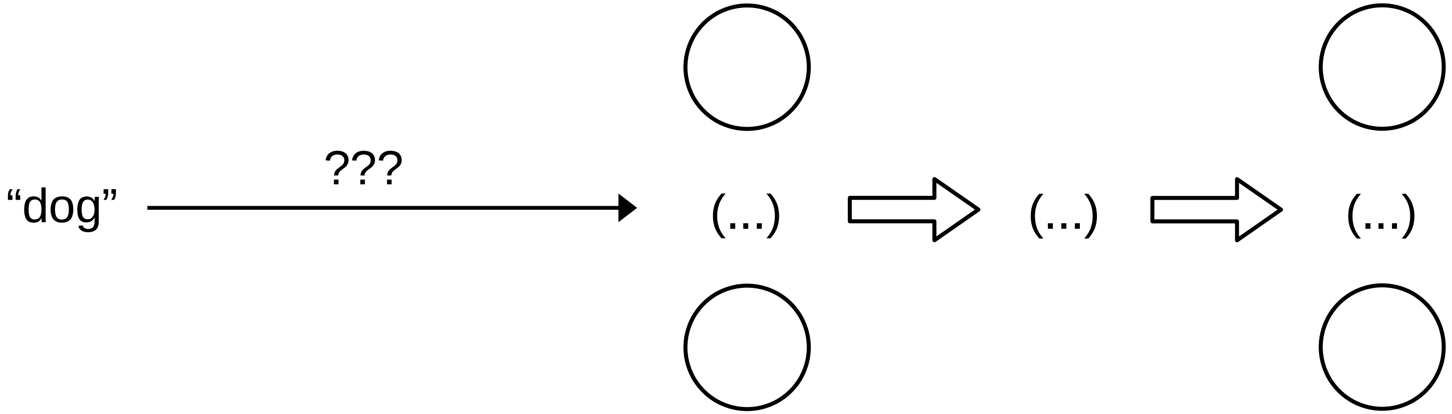


Gradient descent



https://medium.com/@jaleeladejumo/gradient-descent-from-scratch-batch-gradient-descent-stochastic-gradient-descent-and-mini-batch-def681187473

How to get a vector from a word?



How to get a vector from a word?

One-hot encoding

- Vector size = input vocabulary size
- Each word corresponds to a specific vector component
- Representation of word for component *i:* every component is 0 except *i,* which is 1

(0, 0, … 1, … 0, 0)

"dog"

How to get a vector from a word?

Word embeddings

- Dense, real-valued vectors
- Embedding matrix: row for each word's embedding (number of rows = vocabulary size)
- Multiplying one-hot vector with embedding matrix → embedding for the word (table lookup)

Words:	One-hot vectors:
"dog"	[1, 0, 0]
"cat"	[0, 1, 0]
"bird"	[0, 0, 1]

Embedding matrix:

$$\begin{bmatrix} 0.13452 & 0.34562 & 0.65387 & 0.07501 & 0.10053 \\ 0.06753 & 0.28746 & 0.98463 & 0.44763 & 0.00562 \\ 0.34672 & 0.20056 & 0.94461 & 0.37004 & 0.02204 \end{bmatrix}$$

How to get a vector from a word?

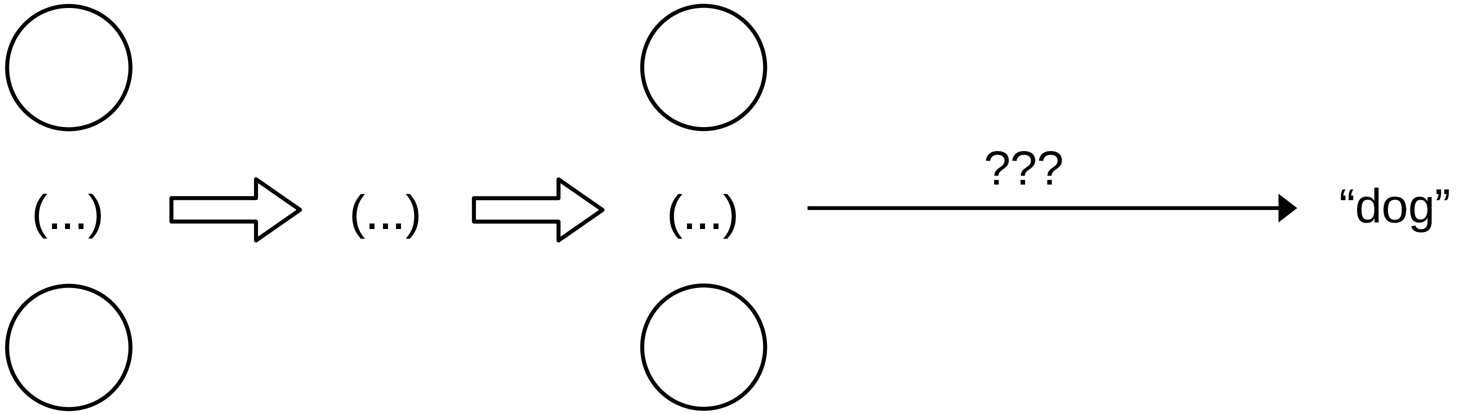
Word embeddings

- Dense, real-valued vectors
- Embedding matrix: row for each word's embedding (number of rows = vocabulary size)
- Multiplying one-hot vector with embedding matrix → embedding for the word (table lookup)

Attaining word embeddings

- Initial layer of LLM: embedding layer
- Pre-trained word embeddings: Word2Vec, GloVe

How to get a word from a vector?



How to get a word from a vector?

Final output: probability distribution across target vocabulary

- Probability distribution: vector of numbers that sum up to 1
- Final layer size = target vocabulary size
- Output is obtained by running final layer through the *softmax* algorithm

$$\textit{softmax}\left(\mathbf{x}\right)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Recurrent Neural Networks (RNNs)

Sequential data: influence of context

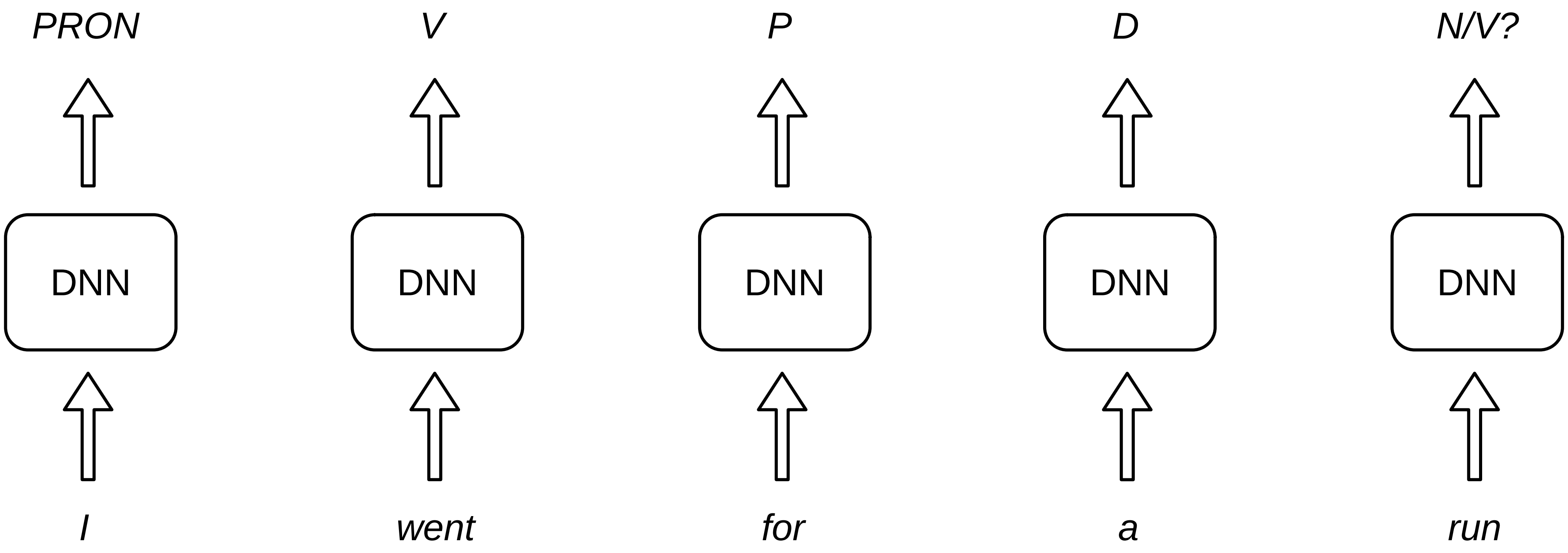
*I **run***

*I went for a **run***

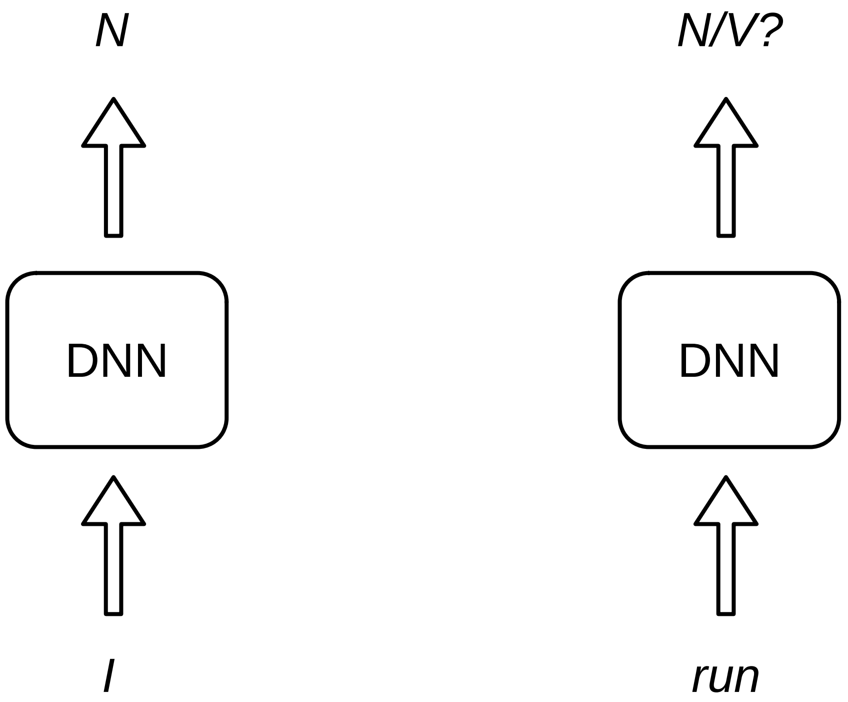
Sequential data: influence of context

I ***run***

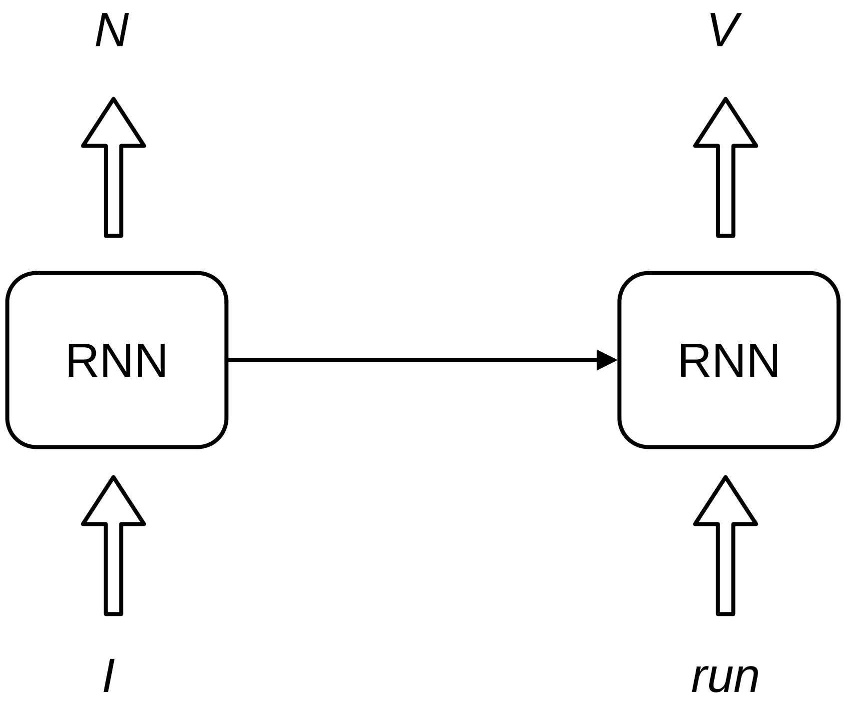
I went for a ***run***



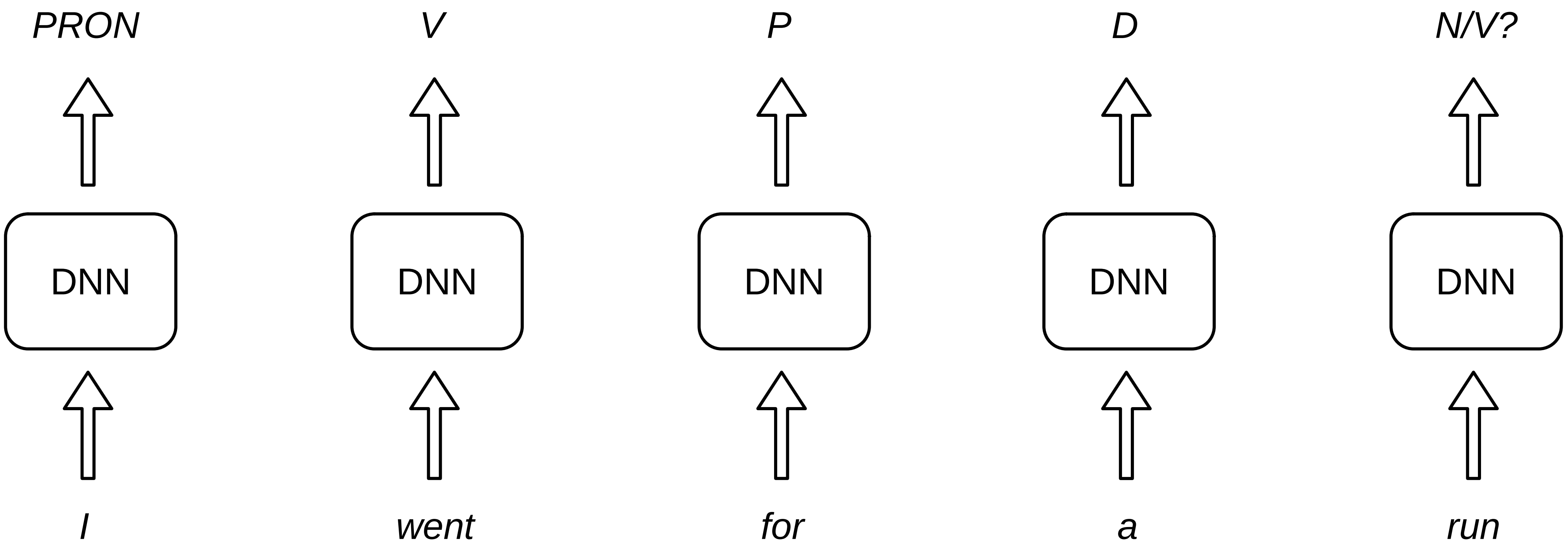
Recurrent connections



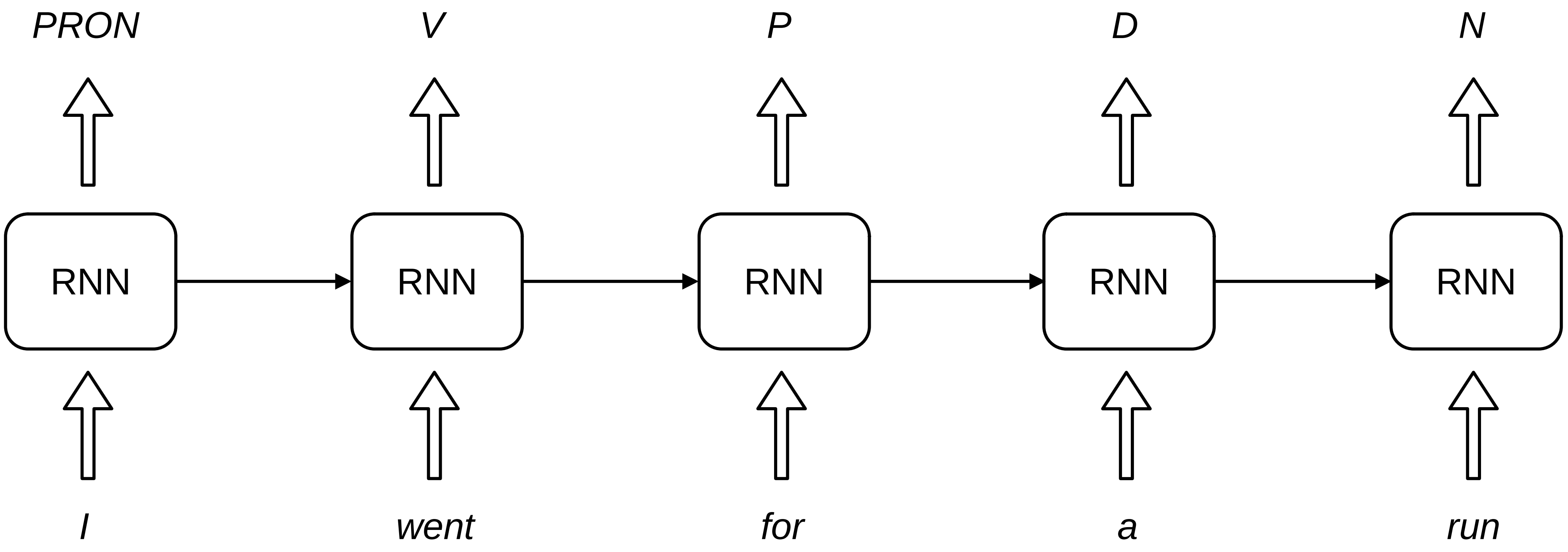
Recurrent connections



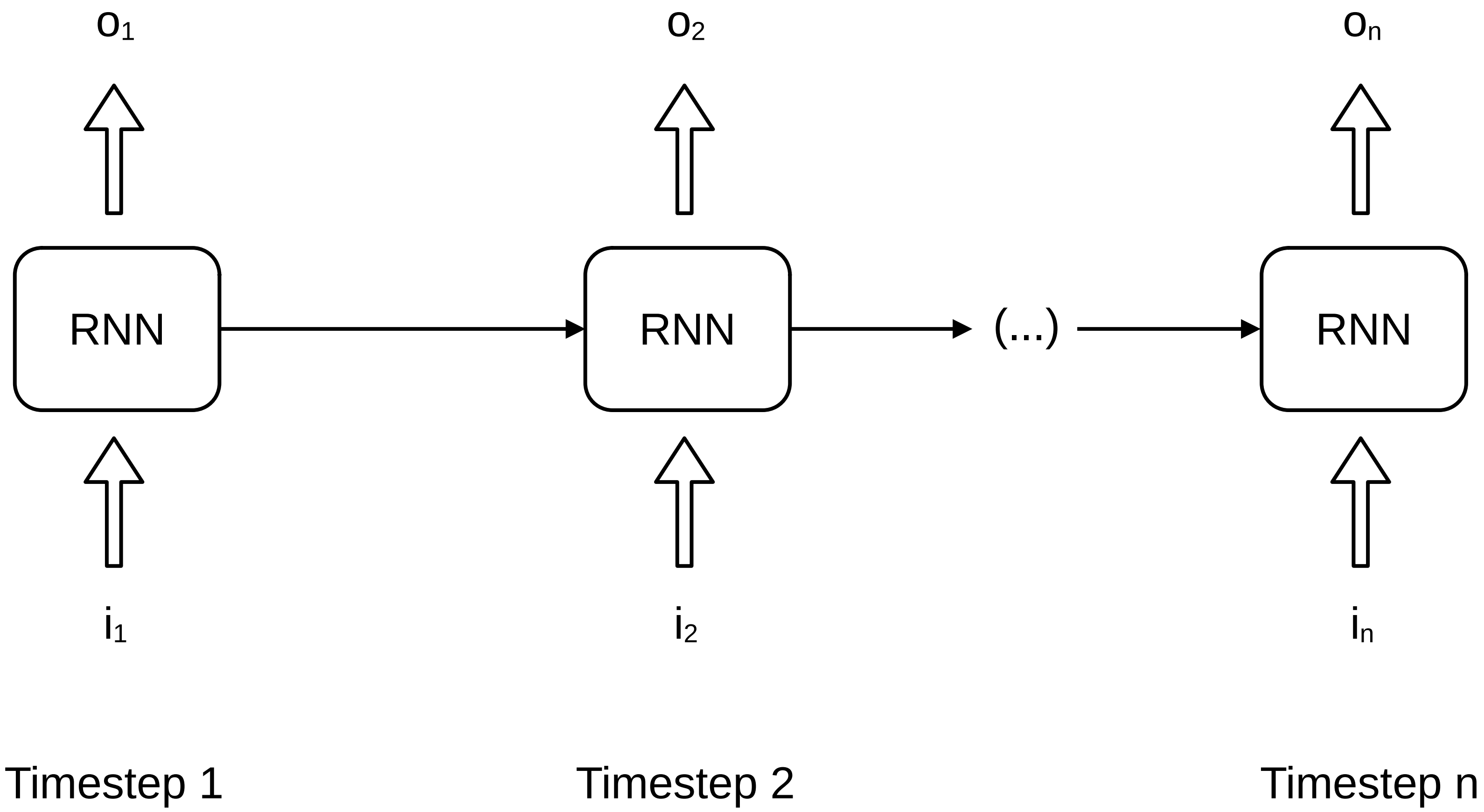
Recurrent connections



Recurrent connections



Recurrent Neural Network (RNN)



Encoder-decoder RNN

Basic RNN maps inputs to outputs 1–1

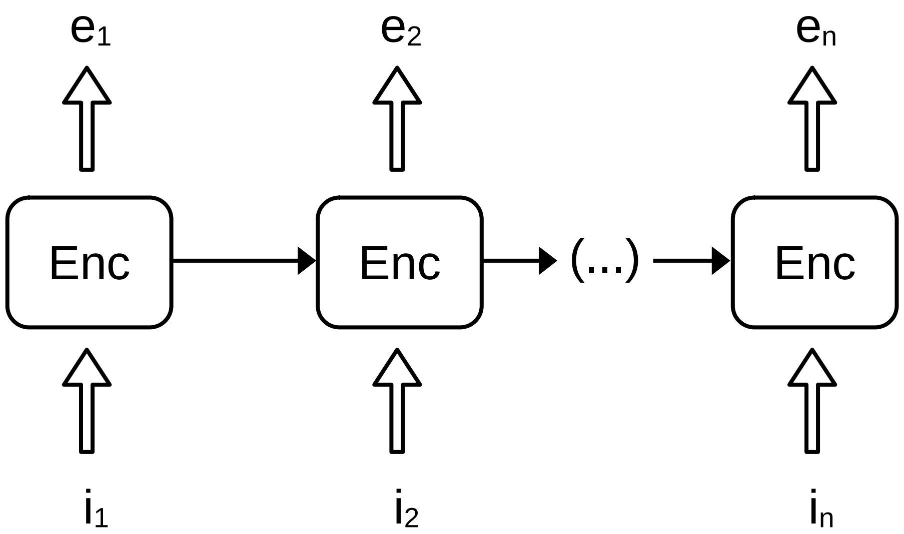
- Part-of-speech tagging
- Spelling correction
- (...)

But we often want more flexible mappings

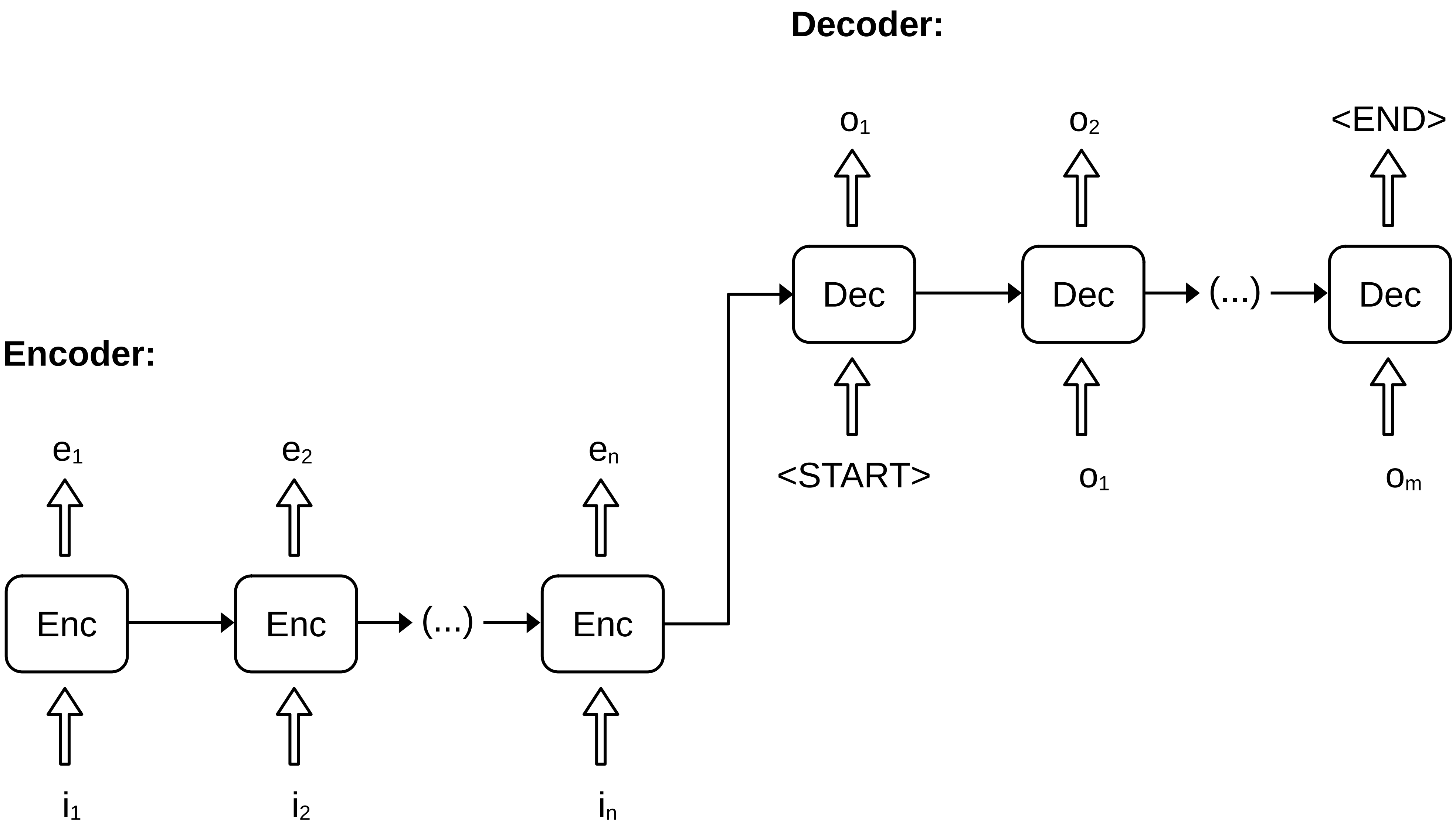
- Machine translation: grammatical and lexical variation between input and output language
- Chatbots: different lengths between prompt and answer
- (...)

Encoder-decoder RNN

Encoder:



Encoder-decoder RNN



Vanishing gradient

Problem

- Older encoder inputs have less effect than more recent ones
- Harder to find long-distance dependencies

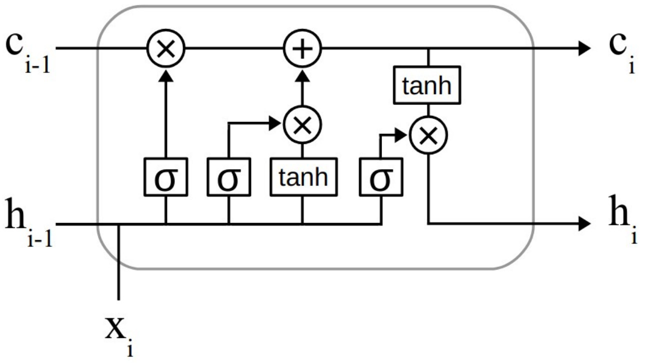
*The **dog** that chased two cats **is** brown*

Vanishing gradient

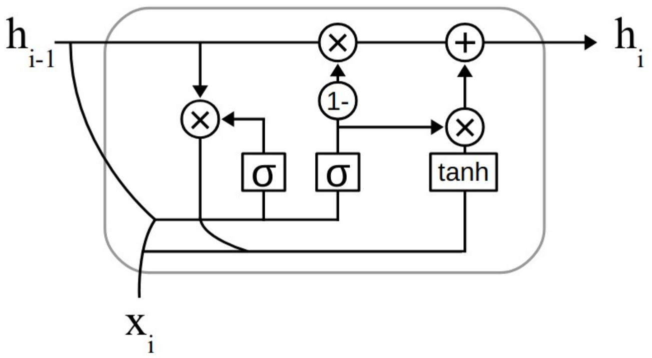
Long short-term memory (LSTM)

- More complex RNN to alleviate the vanishing gradient problem
- Two distinct hidden states updated differently, allowing better retention of information
- *Bidirectional* LSTMs*:* reading input from front-to-back and back-to-front, combining results
- Gated recurrent unit (GRU): similar to LSTM but simpler

LSTM:



GRU:

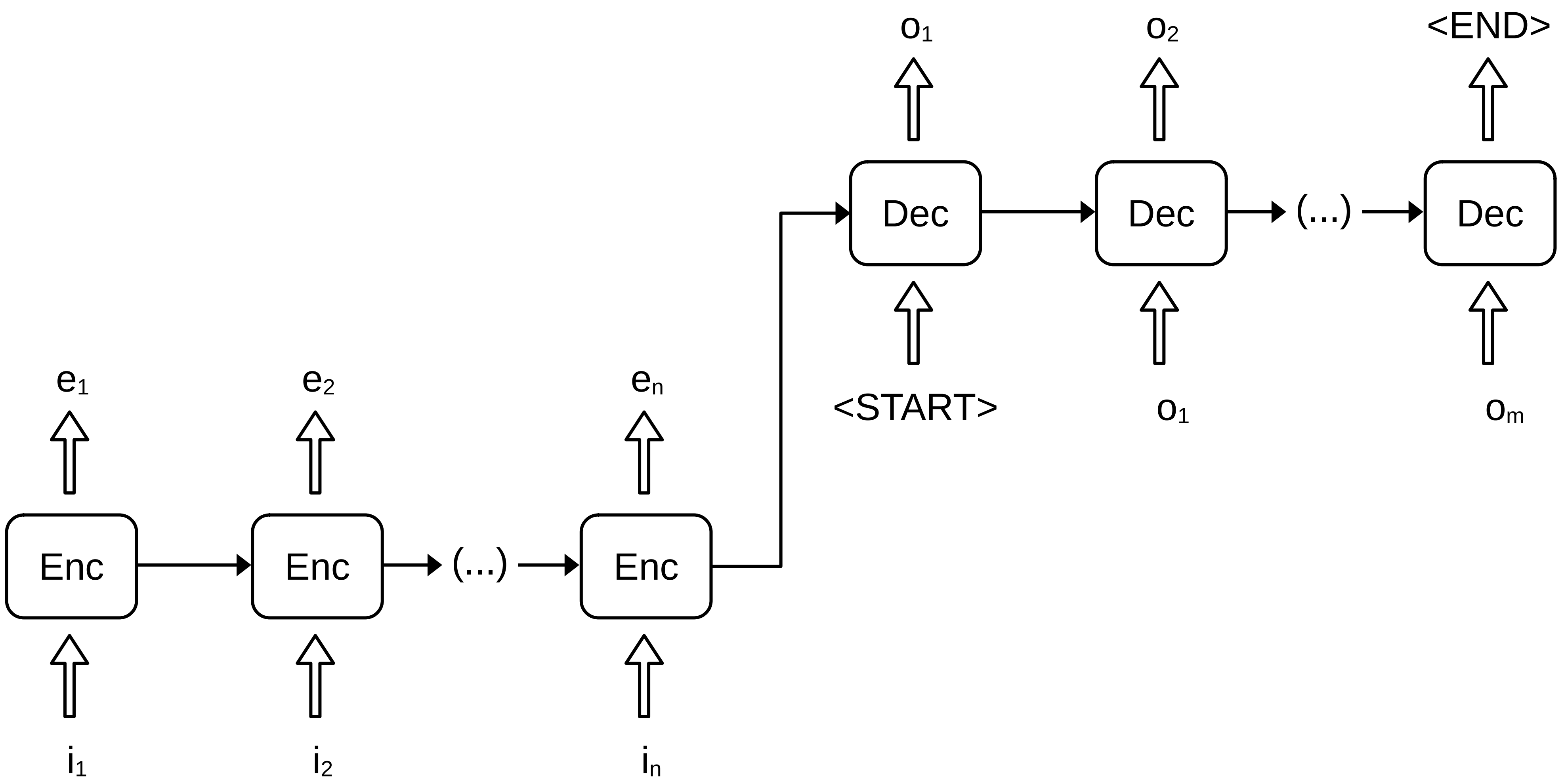


Vanishing gradient

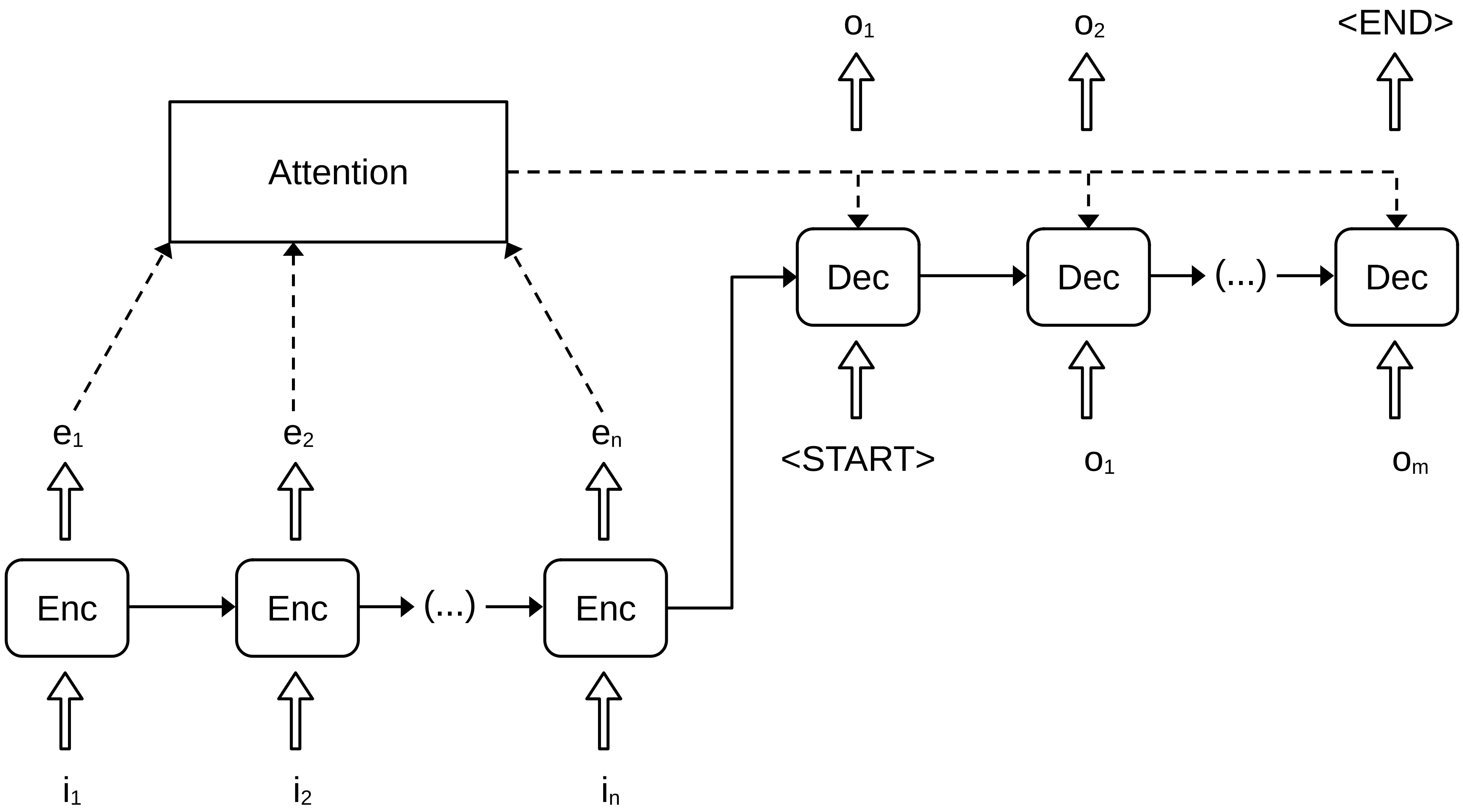
Attention (Bahdanau et al. 2015)

- Calculates a probability distribution across all encoding steps
- Combines all encoder outputs weighted by the probability distribution
- Uses the result as additional decoder input

Encoder-decoder RNN



Encoder-decoder RNN + Attention



Transformer

Transformer

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

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

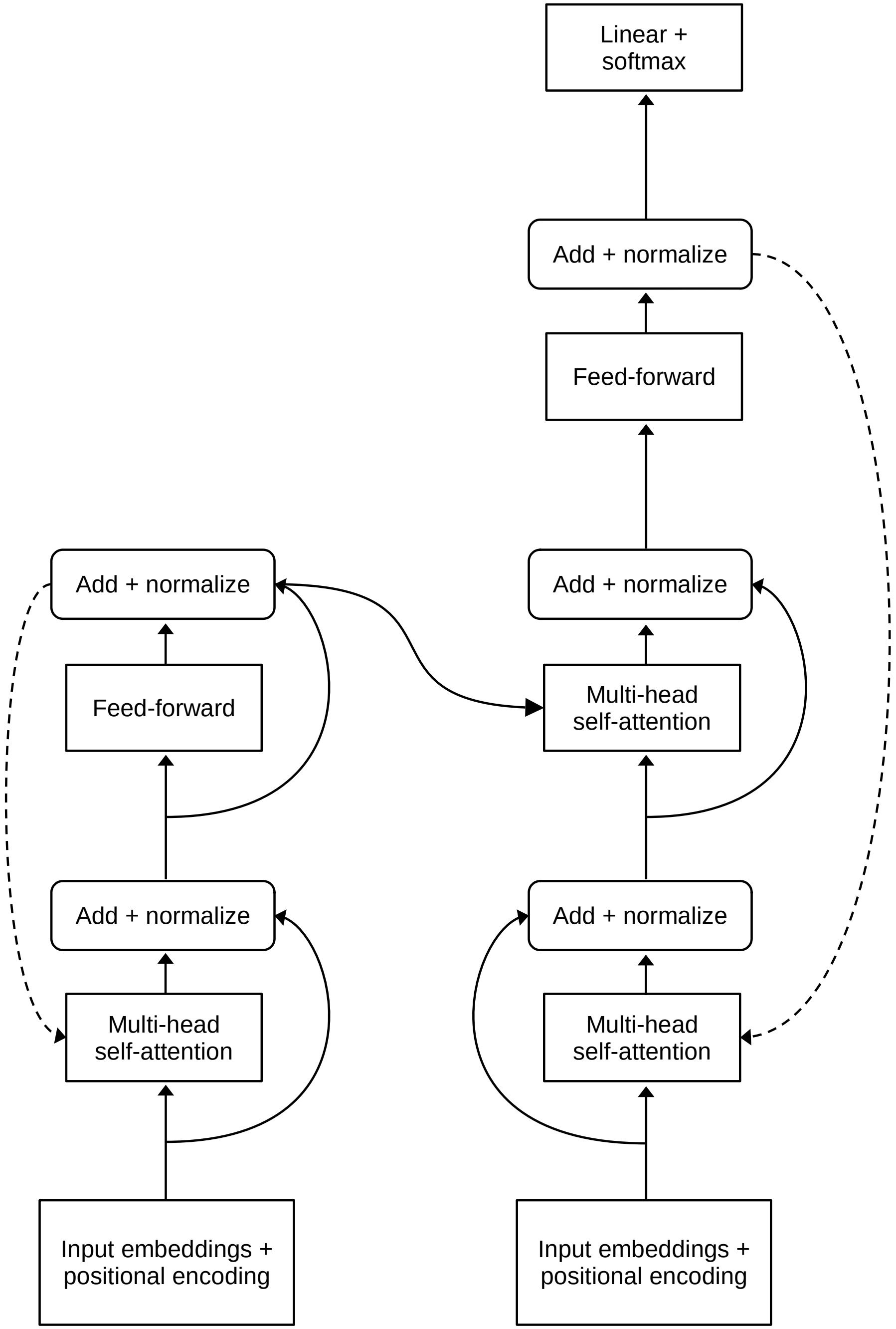
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

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

Transformer



Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

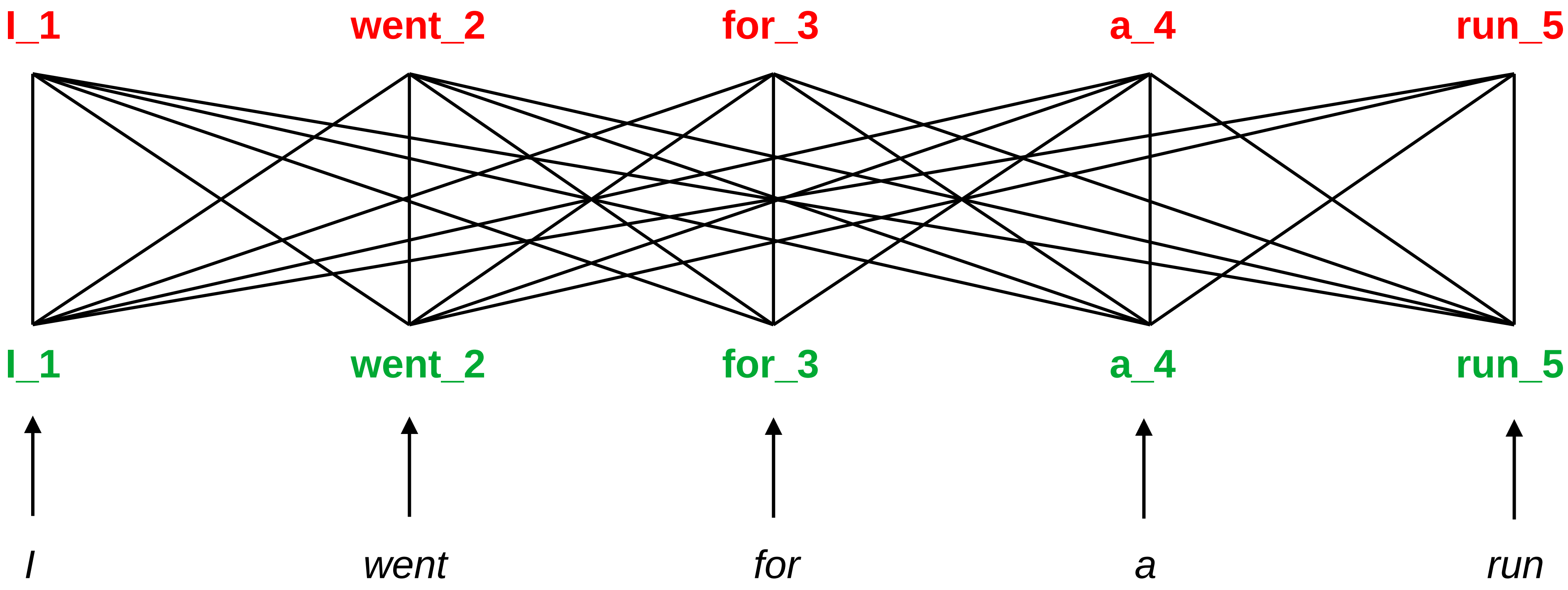
I_1	went_2	for_3	a_4	run_5
↑	↑	↑	↑	↑
I	*went*	*for*	*a*	*run*

Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Input goes through **multi-head self-attention**, creating new **contextual encodings** for each token.

Contextual encoding for each token is calculated from previous embeddings of each token.



Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Input embeddings + positional encoding

Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

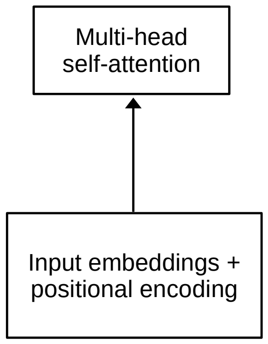
Each Transformer layer contains (several) **attention heads.**

An attention head contains three weight matrices:

query weights: W_q
key weigths: W_k
value weights: W_v

Input embedding x_i is multiplied by each matrix, which yields:

query-vector: $q_i = x_i W_q$
key-vector: $k_i = x_i W_k$
value-vector: $v_i = x_i W_v$



Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Each Transformer layer contains (several) **attention heads.**

An attention head contains three weight matrices:

query weights: W_q
key weigths: W_k
value weights: W_v

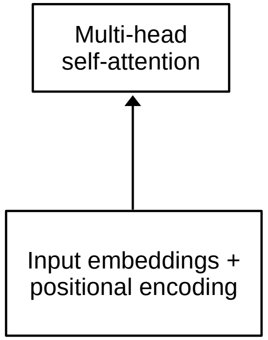
Input embedding x_i is multiplied by each matrix, which yields:

query-vector: $q_i = x_i W_q$
key-vector: $k_i = x_i W_k$
value-vector: $v_i = x_i W_v$

Attention between inputs *i* and *j*:

$$a_{ij} = softmax\left(\frac{q_i . k_j}{\sqrt{d_k}}\right)$$ (d_k = dimensionality of k_j)

Output for input *i* = sum of all v_j weighted with a_{ij}
(**contextual encoding**)



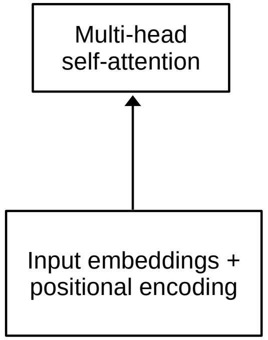
Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Multi-head self-attention:

$$Attention(Q,K,V)=softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Q: query matrix
K: key matrix
V: value matrix

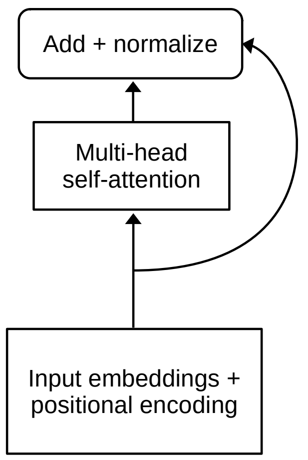


Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Input goes through **multi-head self-attention**.

Outputs of attention heads are combined (+ **residual connections**).



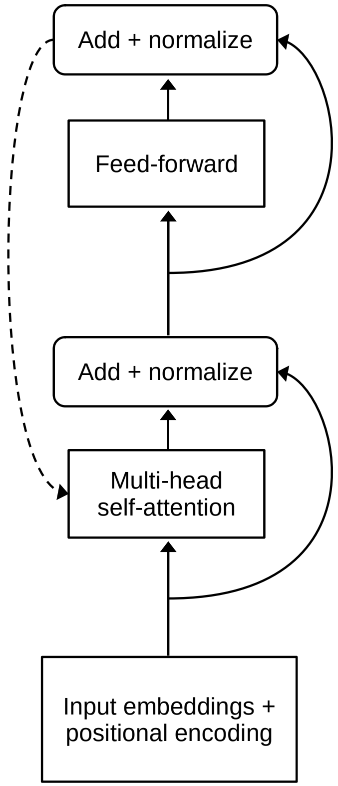
Transformer

Each input word has an **embedding**, which is combined with **positional encoding**.

Input goes through **multi-head self-attention**.

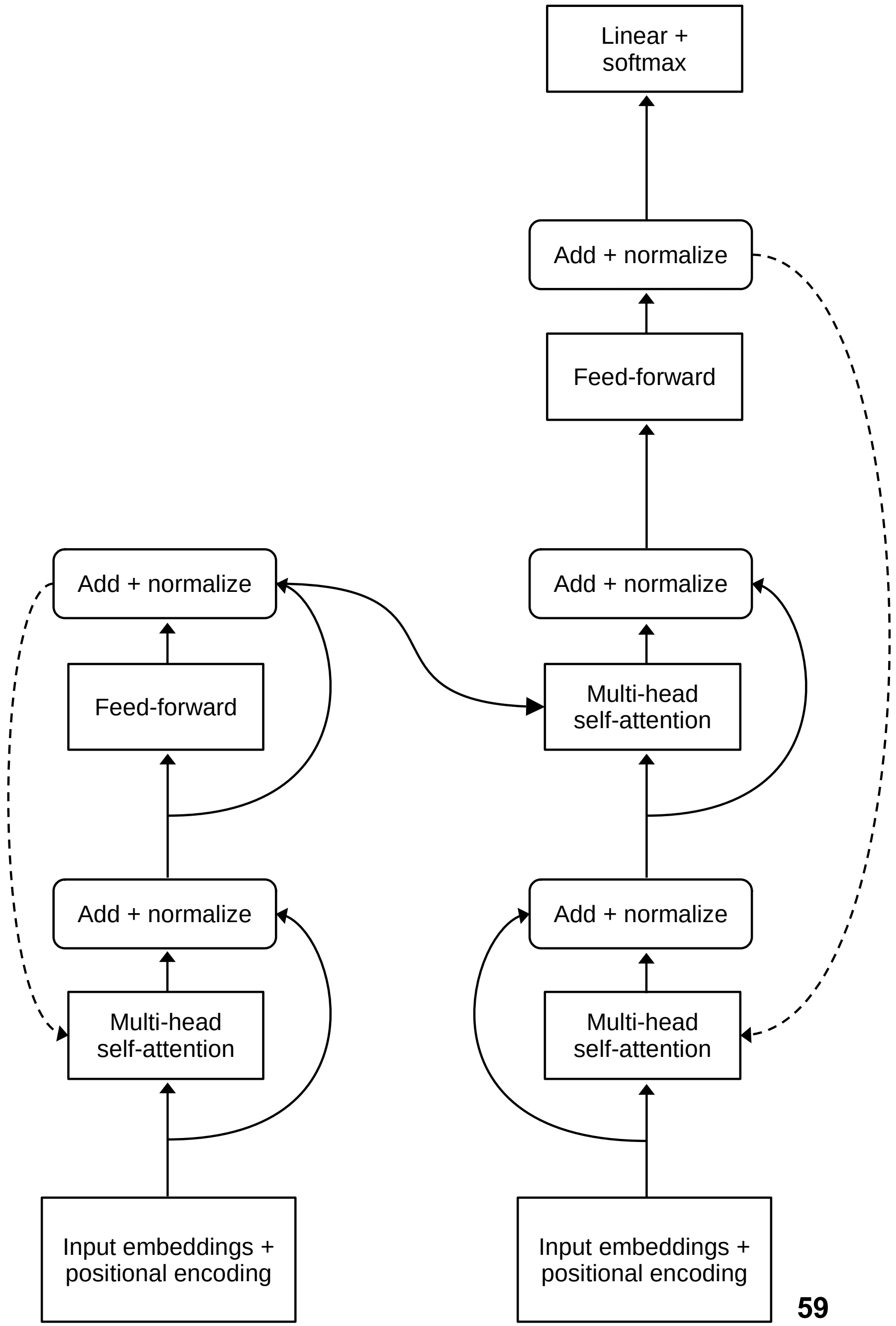
Outputs of attention heads are combined (+ **residual connections**).

Output functions as input to a **feed-forward** network (+ **residual connections**).

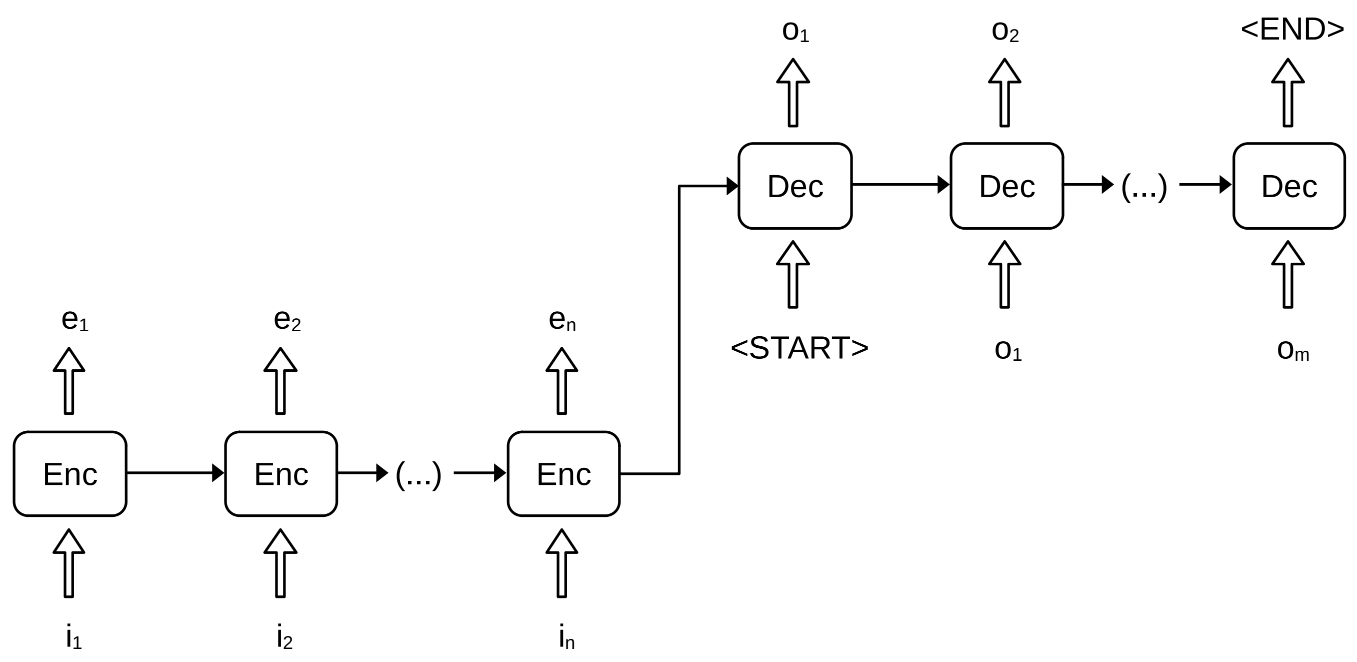


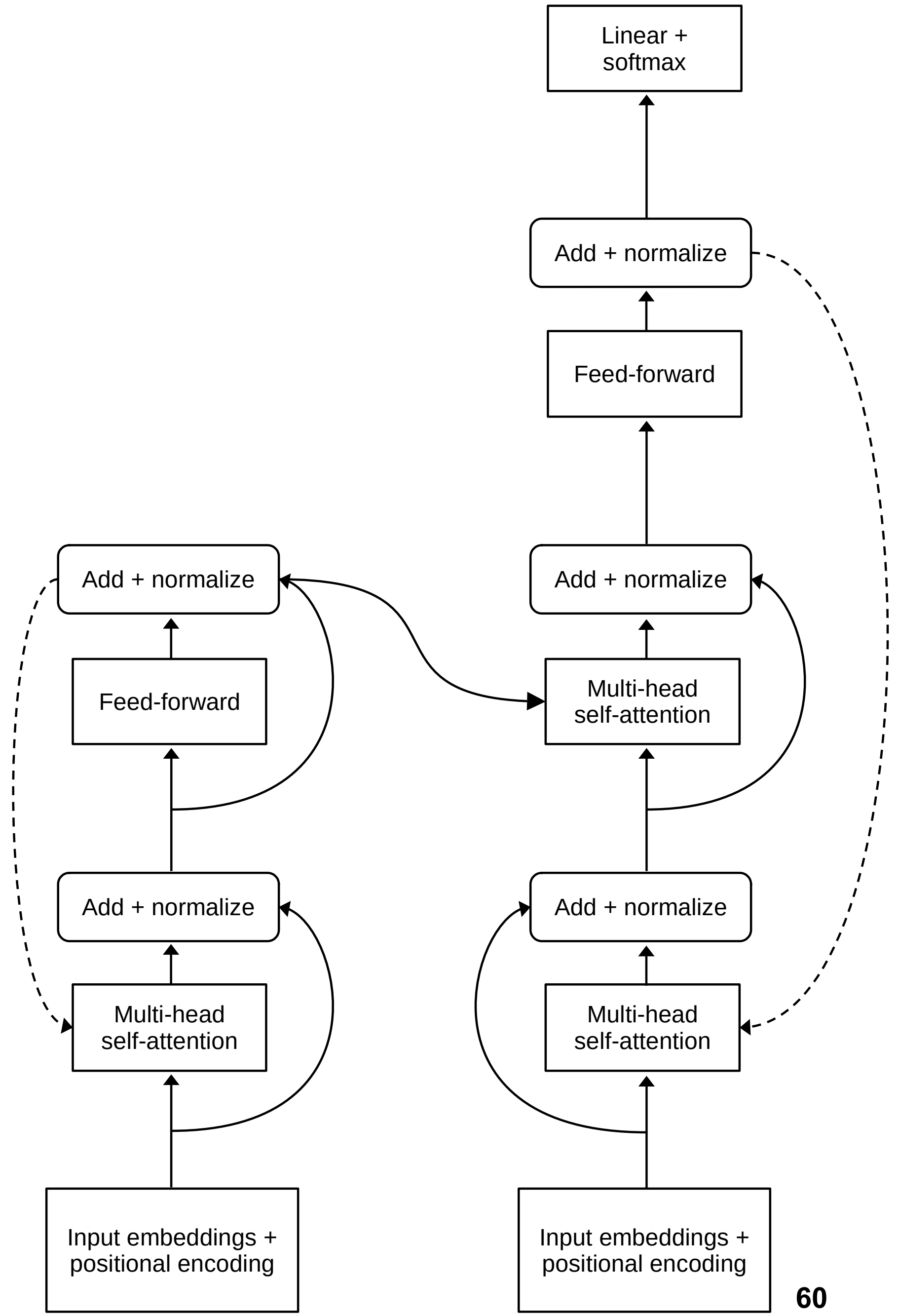
Transformer

Encoder-decoder Transformer: the decoder is like the encoder, but gets additional input via **encoder-decoder attention**



RNN vs. Transformer?



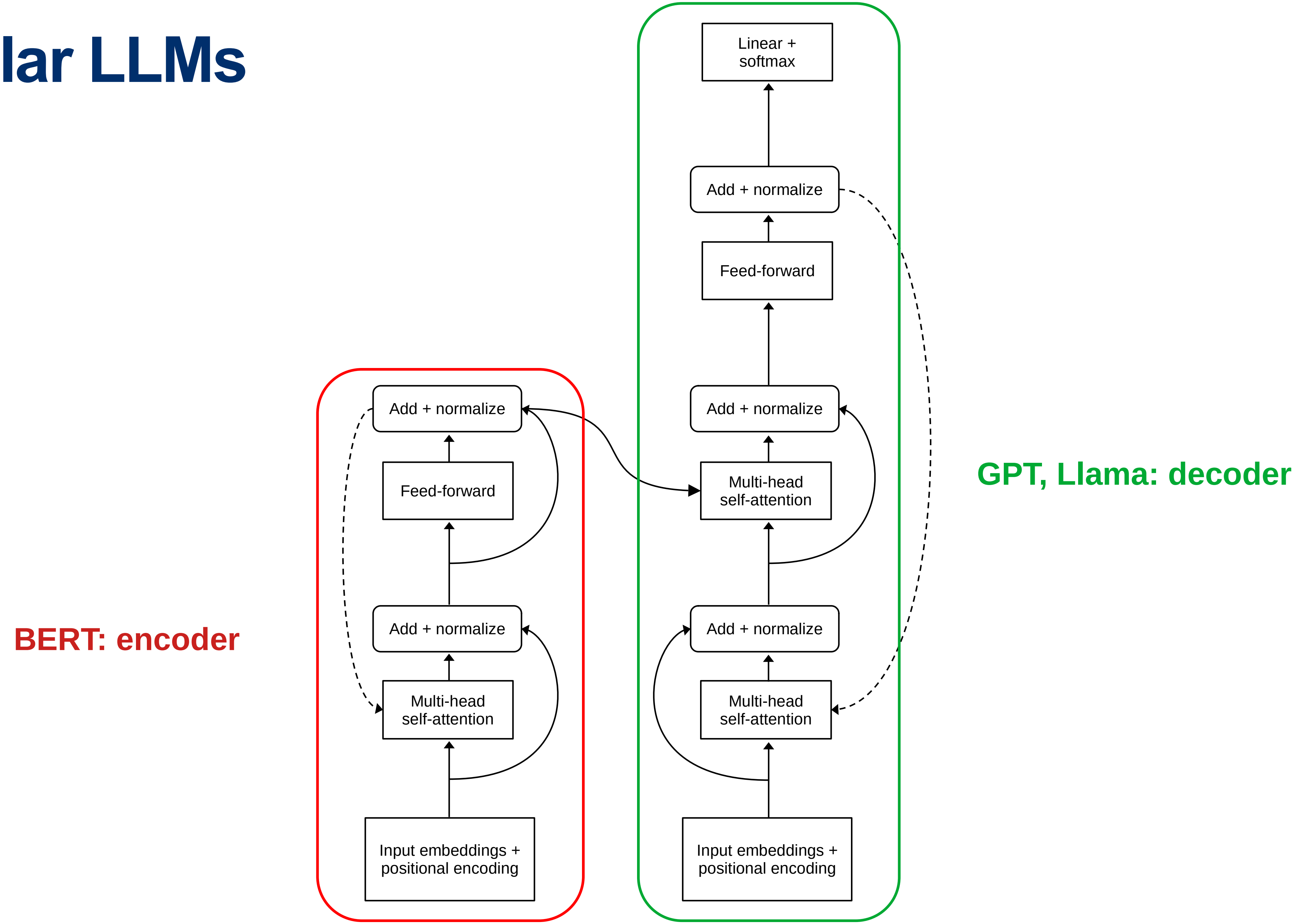


RNN vs. Transformer?

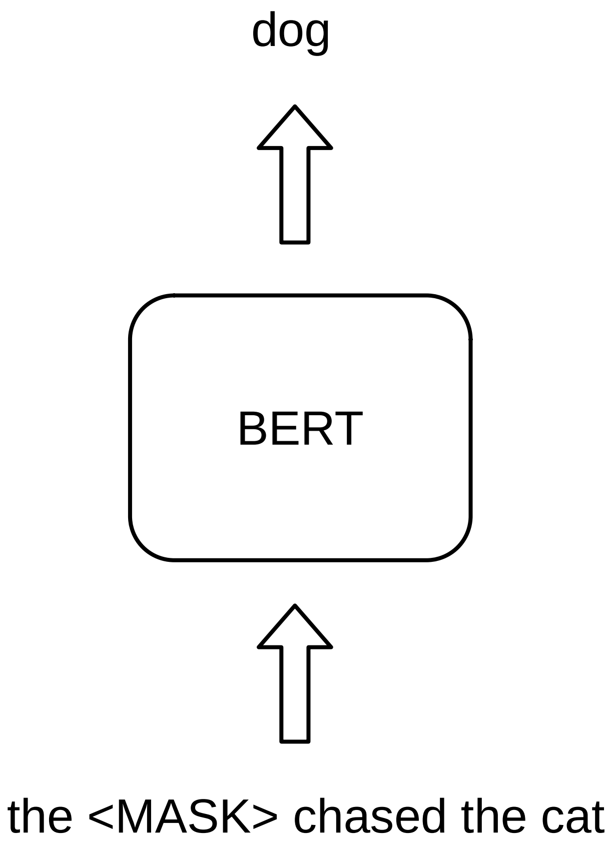
RNN	Transformer
Based on recurrent connections	No recurrent connections
Attention is a useful addition	Fully Attention-based
Goes through the input one token at a time	Goes through all tokens in parallel
Generates one representation of the whole input (last encoding step)	Generates a separate encoding for each input token
Order between tokens arises indirectly via processing steps	Positional encoding added to each input token separately
Long-distance dependencies are especially challenging (vanishing gradient)	Distance between tokens has no direct impact on the strength of their connection

Large Language Models (LLMs)

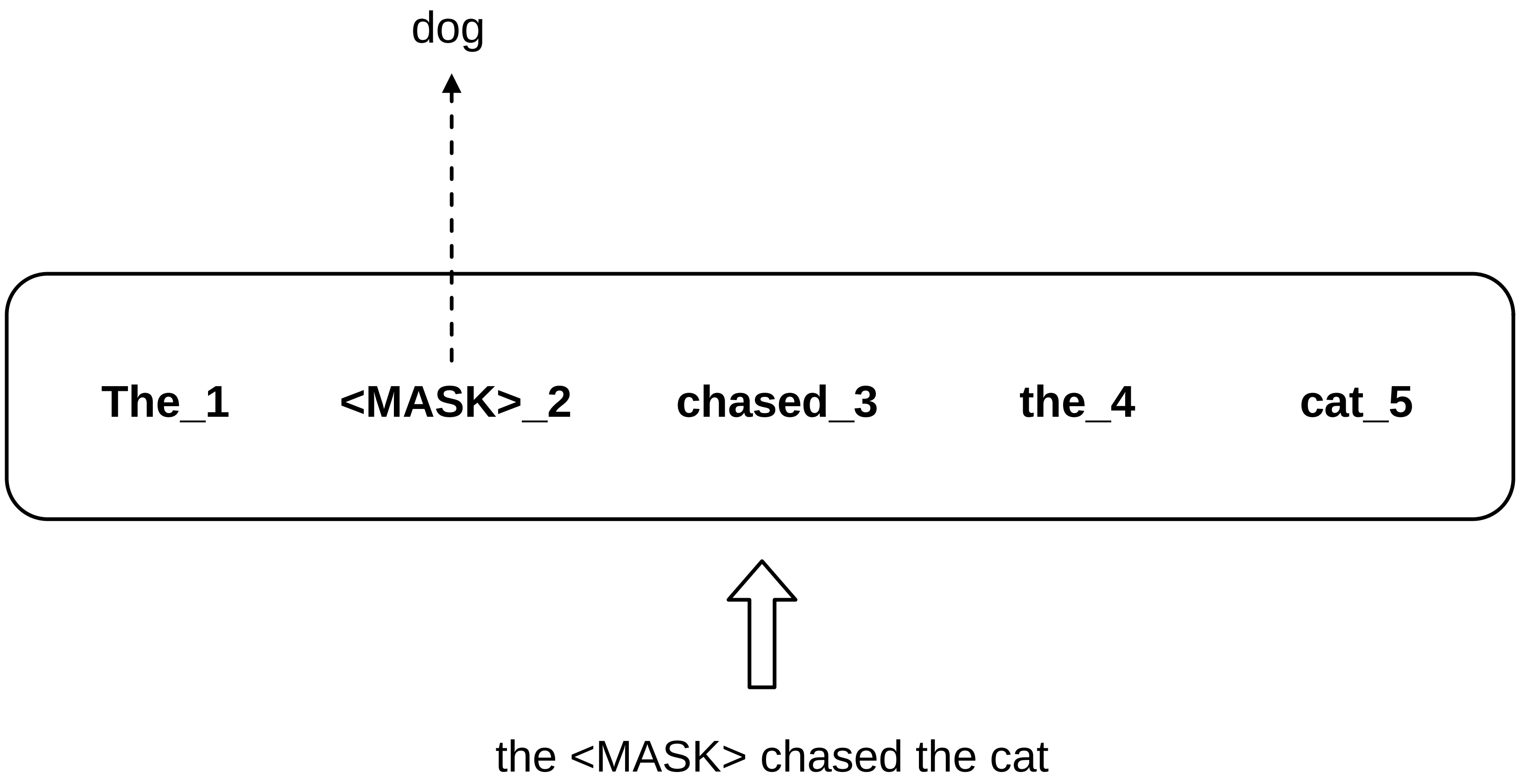
Popular LLMs



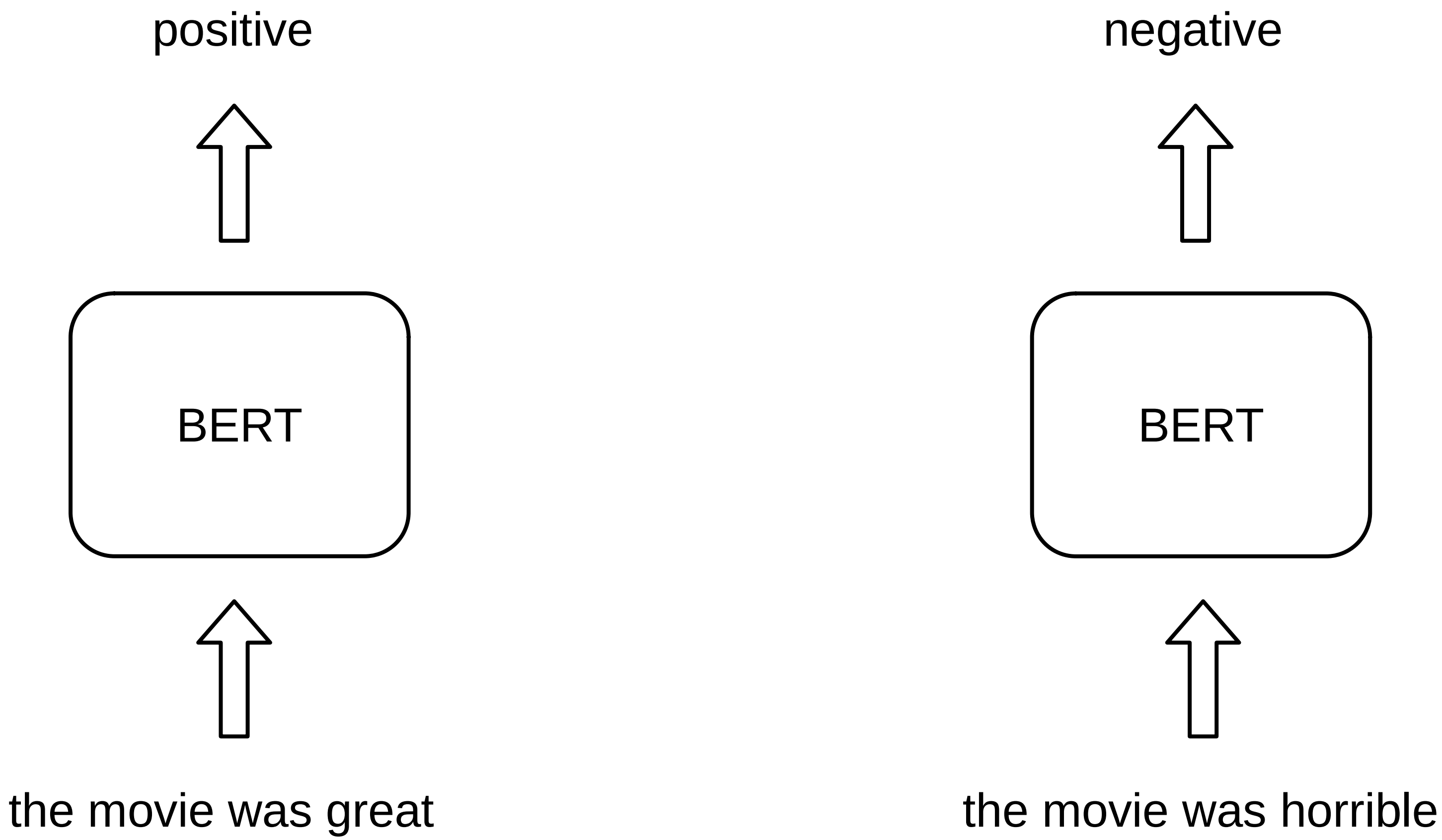
BERT: predicting masked tokens



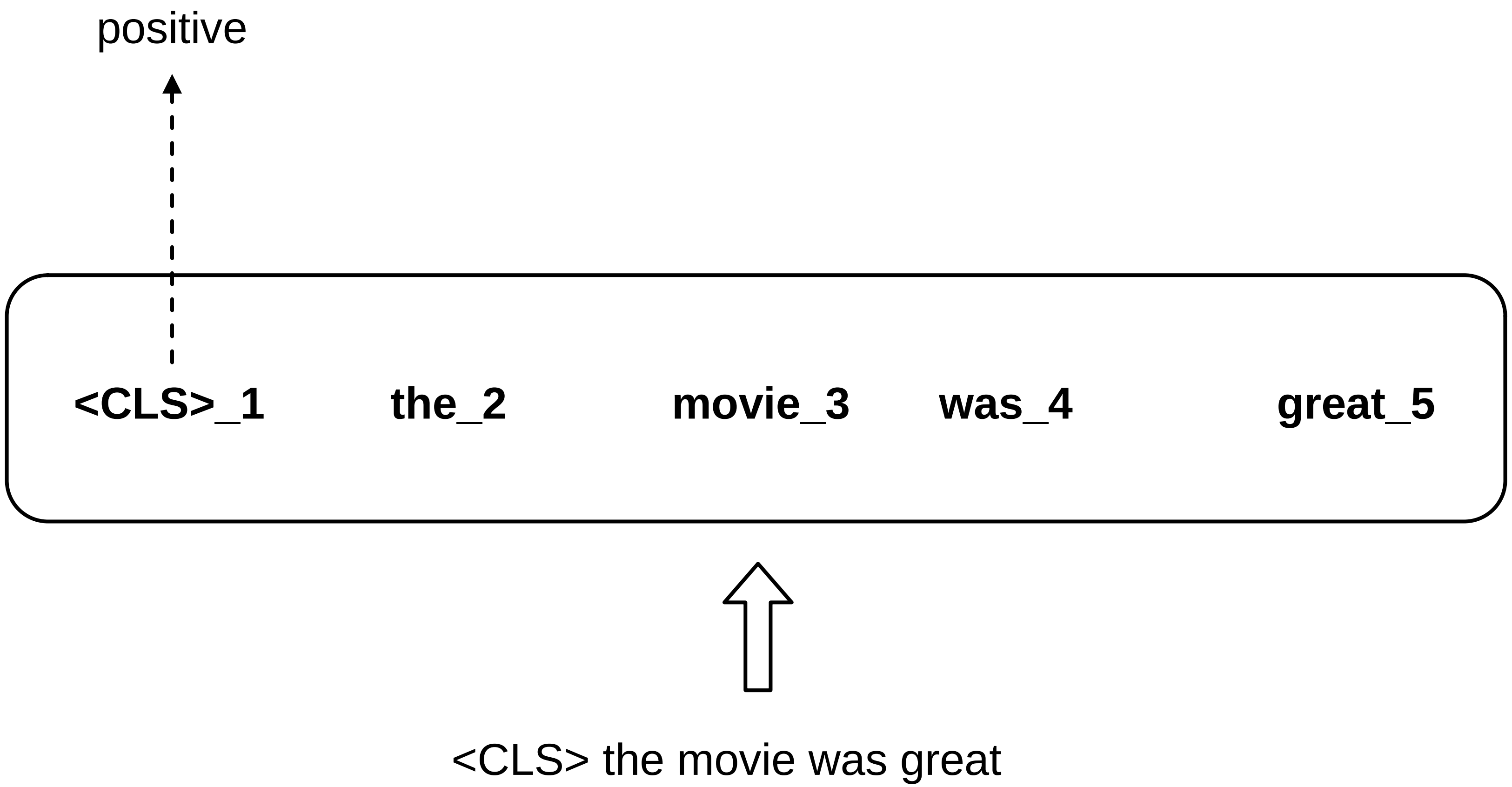
BERT: predicting masked tokens



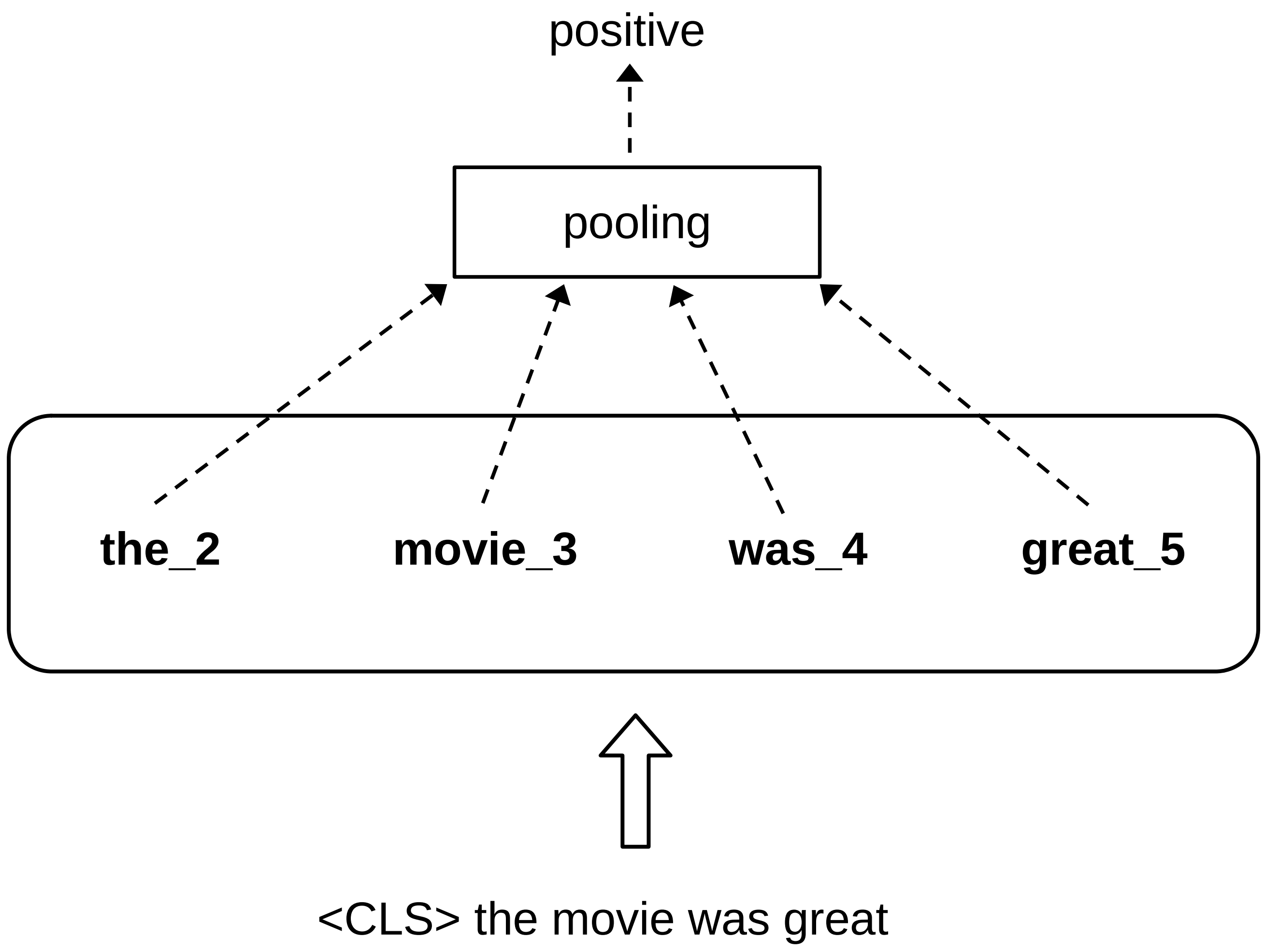
BERT: classifying whole texts



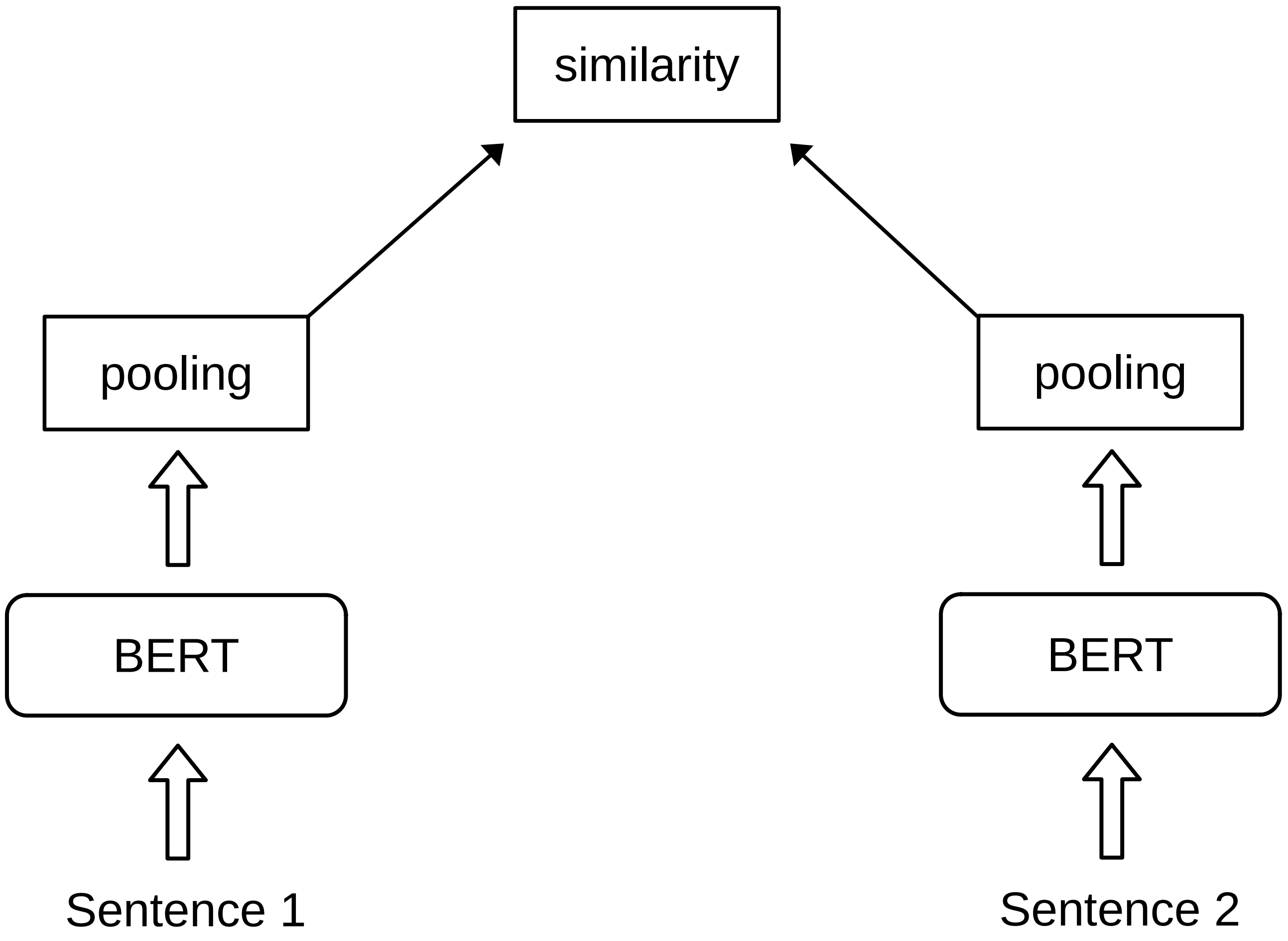
BERT: classifying whole texts via <CLS>-token



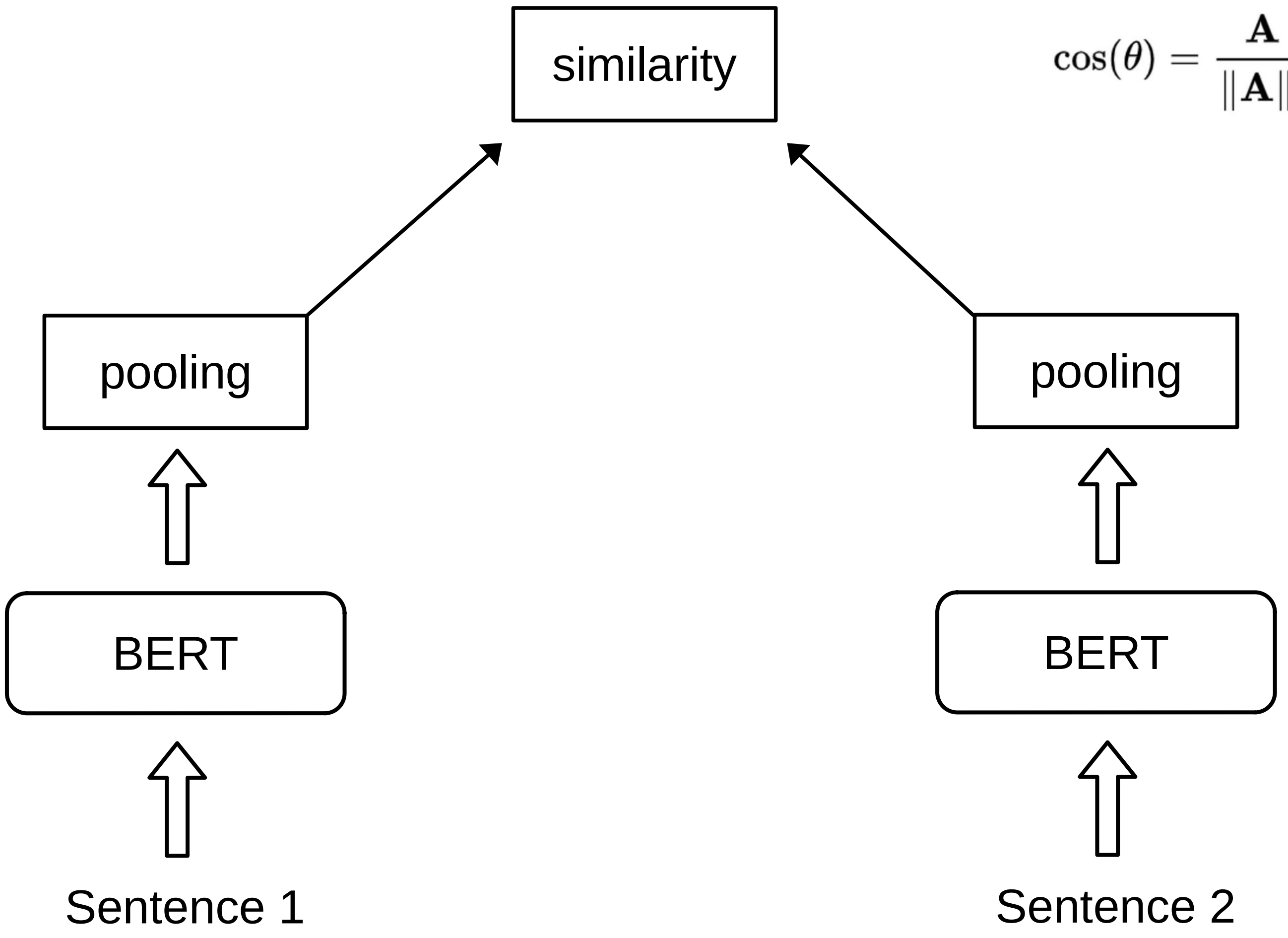
BERT: classifying whole texts via pooling



SentenceBERT: sentence similarities

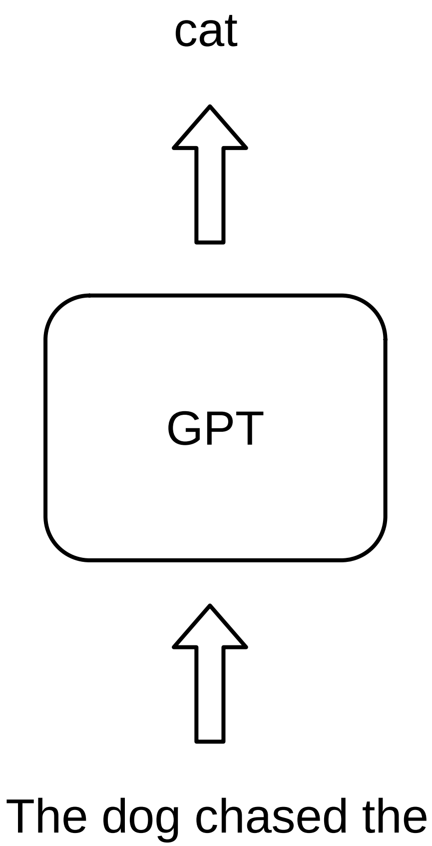


SentenceBERT: sentence similarities

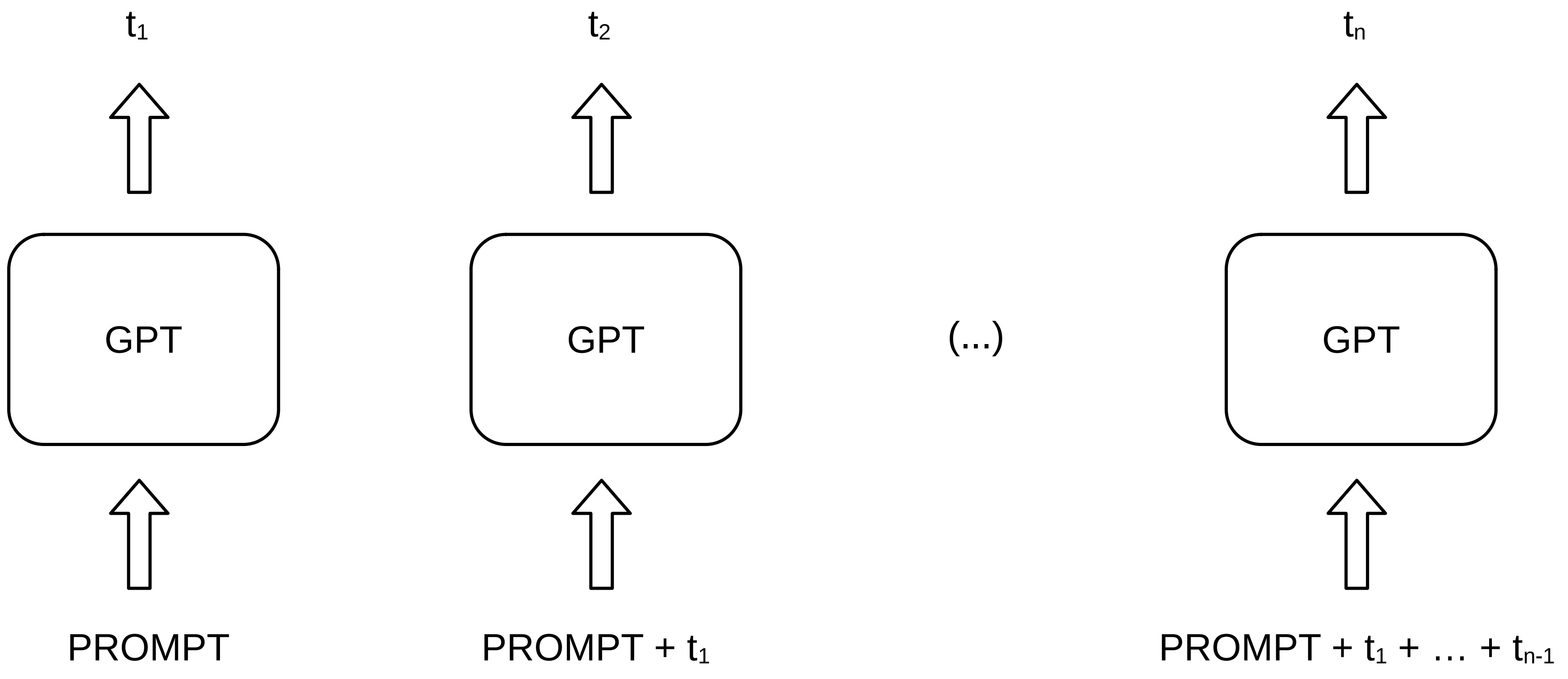


$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

GPT/Llama: predicting the next token



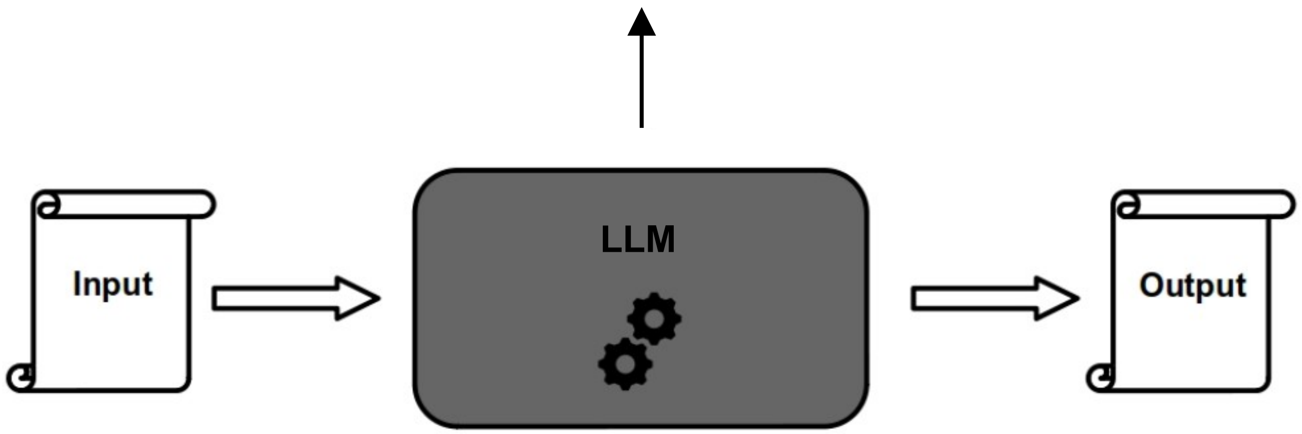
GPT/Llama: predicting the next token



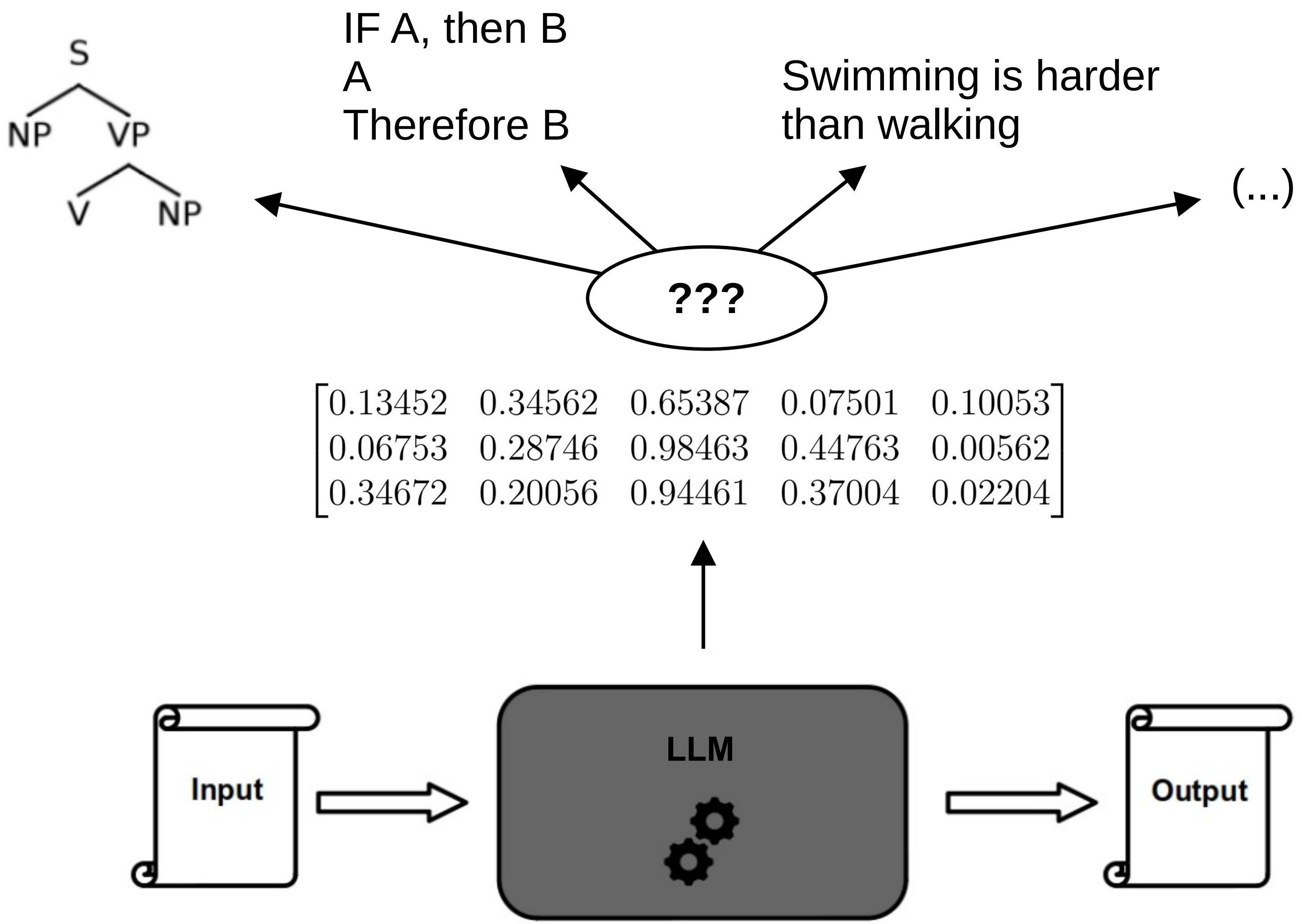
Interpreting LLMs

Challenge

$$\begin{bmatrix} 0.13452 & 0.34562 & 0.65387 & 0.07501 & 0.10053 \\ 0.06753 & 0.28746 & 0.98463 & 0.44763 & 0.00562 \\ 0.34672 & 0.20056 & 0.94461 & 0.37004 & 0.02204 \end{bmatrix}$$



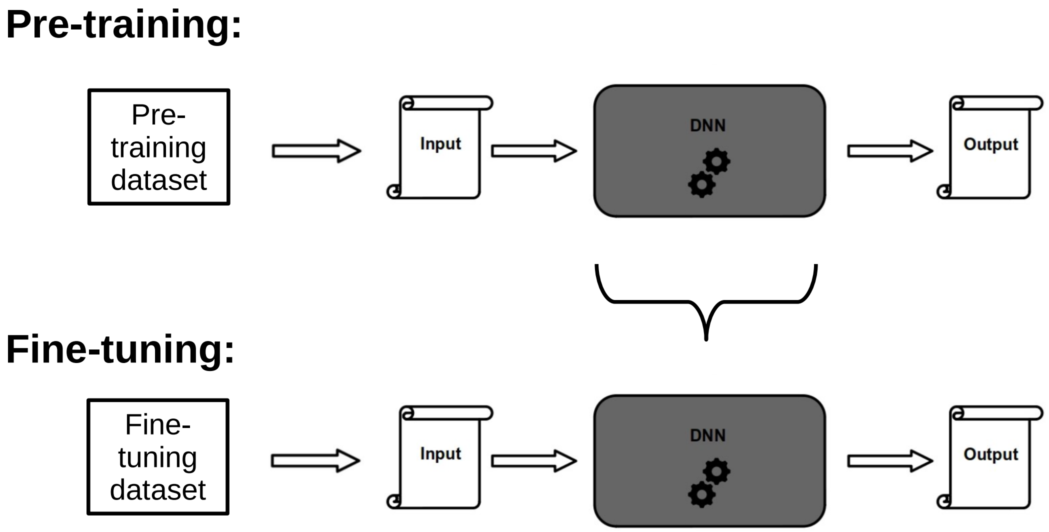
Challenge


S
NP
VP
V
NP
IF A, then B
A
Therefore B
Swimming is harder than walking
(...)
???
0.13452 0.34562 0.65387 0.07501 0.10053
0.06753 0.28746 0.98463 0.44763 0.00562
0.34672 0.20056 0.94461 0.37004 0.02204
Input
LLM
Output

Methods of studying LLMs

Behavioral

- *Fine-tuning* for downstream tasks, measuring performance (*transfer learning*)
- Prompting pre-trained models directly



Now we are going to say which sentences are acceptable (i.e., grammatical) and which are not.

Sentence: Flosa has often seen Marn.
Answer: good

Sentence: Chardon sees often Kuru.
Answer: bad

Sentence: Bob walk.
Answer: bad

Sentence: Malevolent floral candy is delicious.
Answer: good

Sentence: The bone chewed the dog.
Answer: good

Sentence: The bone dog the chewed.
Answer: bad

Sentence: I wonder you ate how much.
Answer: bad

Sentence: The fragrant orangutan sings loudest at Easter.
Answer: good

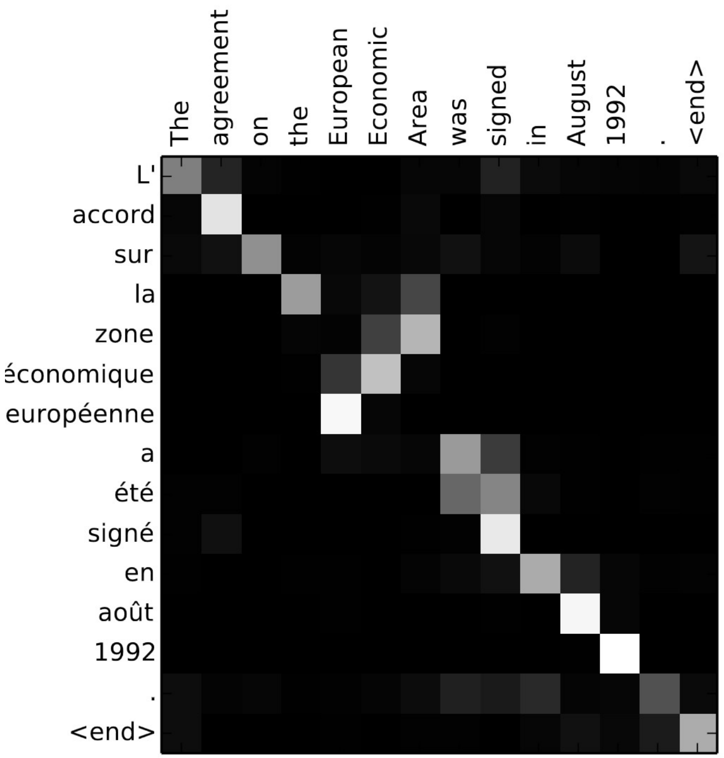
Sentence: [TEST SENTENCE GOES HERE]
Answer:

(Mahowald 2023)

Methods of studying LLMs

Attention visualization

- Displaying how attention is allocated for each contextual encoding

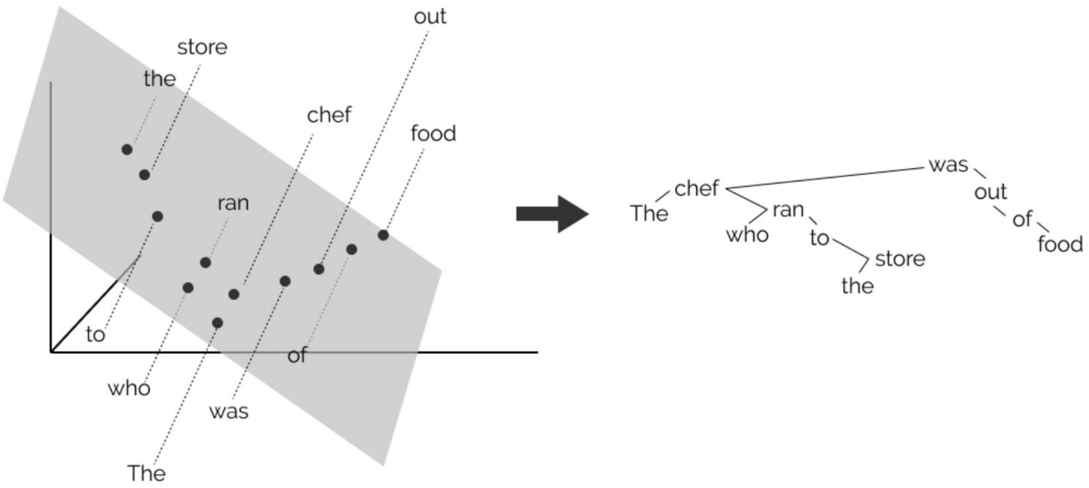


(Bahdanau et al. 2015)

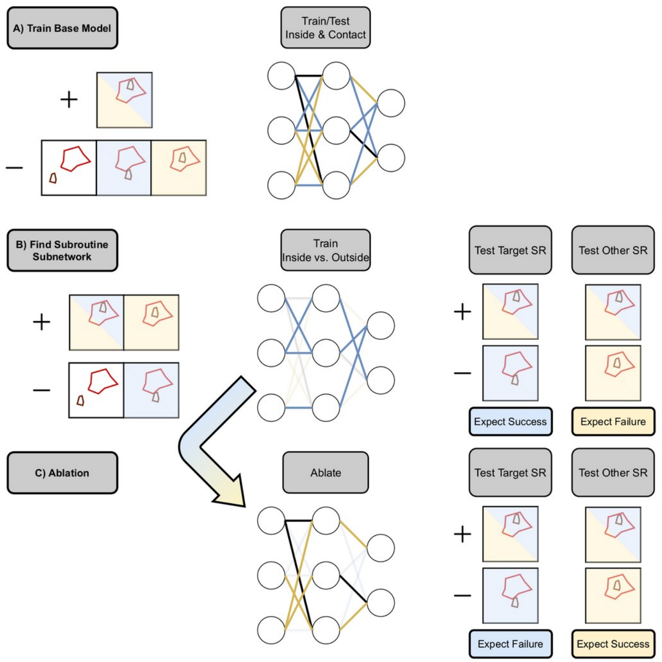
Methods of studying LLMs

Looking inside LLMs

- *Probing:* mapping activation patterns to linguistic/semantic labels
- *Mechanistic interpretation:* opening up the computational pipeline



https://nlp.stanford.edu/~johnhew/structural-probe.html

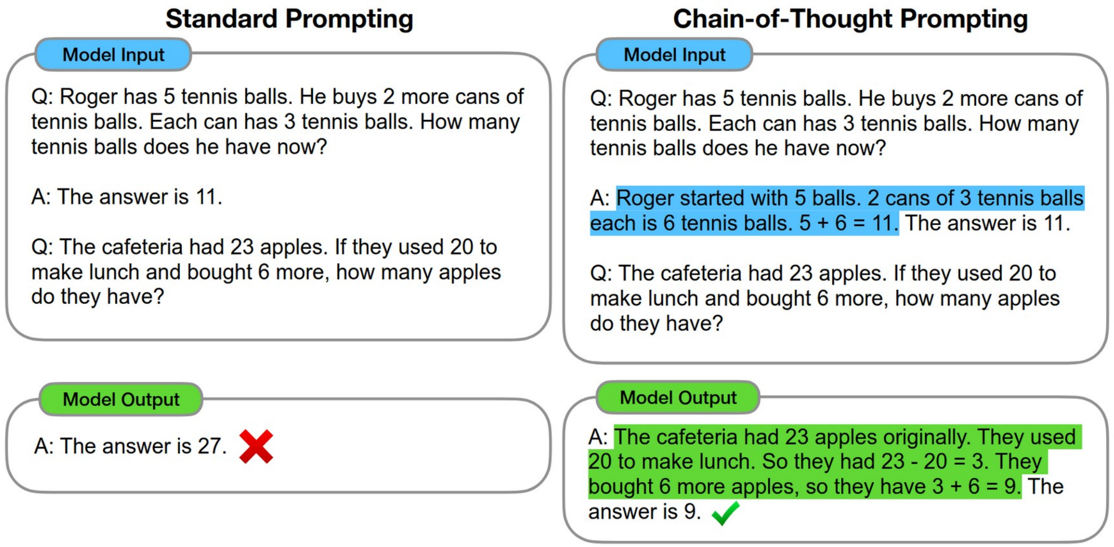


(Lepori et al. 2023)

Methods of studying LLMs

Chain-of-thought reasoning

- Giving the LLM examples of explicit reasoning → facilitating the generation of reasoning steps
- Improves performance in many complex reasoning tasks (Chu et al. 2024)
- Theoretical explanation is an important challenge (Feng et al. 2023, Prystawski et al. 2023)



(Wei et al. 2022)

Questions?

Information about practical session tomorrow

Instructions: https://github.com/tombgro/llm-workshop

- Jupyter notebook using Python 3.12
- Install Anaconda/Miniconda create virtual environment (highly recommended)
- Install libraries, decide on GPU/CPU-version of Pytorch
- To see if you have CUDA installed, run this in terminal (command line): nvcc --version

References

Dzmitry Bahdanau, Kyunghyun Cho, & Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. Proceedings of the International Conference on Learning Representations (ICLR).

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, & Ting Liu. 2024. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1173–1203.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: A theoretical perspective. Thirty-seventh Conference on Neural Information Processing Systems, NeurIPS.

Michael Lepori, Thomas Serre, & Ellie Pavlick. 2023. Break it down: Evidence for structural compositionality in neural networks. Advances in Neural Information Processing Systems 36: 42623-42660.

Kyle Mahowald. 2023 A Discerning Several Thousand Judgments: GPT-3 Rates the Article + Adjective + Numeral + Noun Construction. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics.

Ben Prystawski, Michael Li, and Noah D. Goodman. 2023. Why think step by step? Reasoning emerges from the locality of experience. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, & Illia Polosukhins. 2017. Attention is all you need. Proceedings of the 31st International Conference on Neural Information Processing, pages 6000–6010.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, & Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems 35.