NLP and the Web – WS 2024/2025

Lecture 9
Neural Language Modeling

Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

Syllabus (tentative)

Nr.	Lecture
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	IR – Word Representation
05	IR – Transformer/BERT
06	IR – Dense Retrieval
07	IR – Neural Re-Ranking
08	LLM – Language Modeling Foundations, Tokenization
09	**LLM – Neural LLM**
10	LLM – Adaptation, LoRa, Prompting
11	LLM – Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	LLM – Scaling, Computation Cost
14	Review & Preparation for the Exam

Outline

Recap: LM and Sub-word tokenization

Basic Neural Language Models

RNN and Transformer LM

Recap: Language Models

Language Model (LM) = Model that assigns probabilities to sequences of words

Recap: Bigram LM

Word 1	Word 2	Count	P(W2|W1)
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
<S>	I	69	0.069
<S>	They	169	0.169
<S>	Also	123	0.123
<S>	However	45	0.045
Anything	else	…	

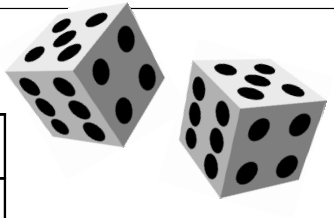
$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}\, w_n)}{C(w_{n-1})}$$

$$C(<S>) = 1000$$

$$\frac{C(<S> There)}{C(<S>)} = \frac{256}{1000} = 0.256$$

Recap: Generation with a Bigram LM

Generate the next token based on conditional probabilities



There...

Word 1	Word 2	Count	P(W2|W1)
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
…	…	…	…
There	**once**	**123**	**0.0246**
There	**must**	**29**	**0.0058**
There	**has**	**69**	**0.0138**
…	…	…	…

There once

Recap: OOV problem

Big problem: zero probability n-grams

In Evaluation: Test set might contain N-Grams that did never appear in training
= the entire probability of the test set is zero! (multiplication with zero)
= Perplexity is undefined! (division by zero)

In Generation: LM has never seen the words of my prompt before
= No way to generate the next token!

Approaches: <UNK> tokens, Laplace or add-k smoothing, backoff, interpolation…

Recap: Byte Pair Encoding (BPE)

Subword tokenization technique

Used for data compression and dealing with unknown words

Initialization:

Vocabulary = set of all individual characters

V = {A, B, C, … a, b, c, … 1, 2, 3, … !, $, %, …}

Repeat:

- Choose two symbols that appear as a pair most frequently (say “a” and “t”)
- Add new merged symbol (“at”)
- Replace each occurrence with the new symbol (“t”,”h”,”a”,”t” -> “t”,”h”,”at”)

Until k merges have been done

De-Tokenization

Greedy longest prefix matching. Example: “the golden snickers”

vocab: {“the”, “go”, “gold”, “##den”, “##en”, “snicker”, “##ers” , “##s”}

the golden snickers -> “**the**” in vocab -> [“the”]

the **golden** snickers -> “golden” not in vocab

-> prefix max match: “**gold**” -> [“the”, “gold”]

-> remaining subword: “##**en**” -> “##en” in vocab -> [“the”, “gold”, “##en”]

the golden **snickers** -> “snickers” not in vocab

-> prefix max match: “**snicker**” in vocab -> [“the”, “gold”, “##en”, “snicker”]

-> remaining subword: “##**s**” in vocab -> [“the”, “gold”, “##en”, “snicker”, “##s”]

GPT3/4's Tokenizer

OpenAI's large language models (sometimes referred to as GPT's) process text using tokens, which are common sequences of characters found in a set of text. The models learn to understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text might be tokenized by a language model, and the total count of tokens in that piece of text.

It's important to note that the exact tokenization process varies between models. Newer models like GPT-3.5 and GPT-4 use a different tokenizer than our legacy GPT-3 and Codex models, and will produce different tokens for the same input text.

Here is a math problem: 234566+64432 / (33345) * 0.1234

https://platform.openai.com/tokenizer

Outline

Recap: LM and Sub-word tokenization

Basic Neural Language Models

RNN and Transformer LM

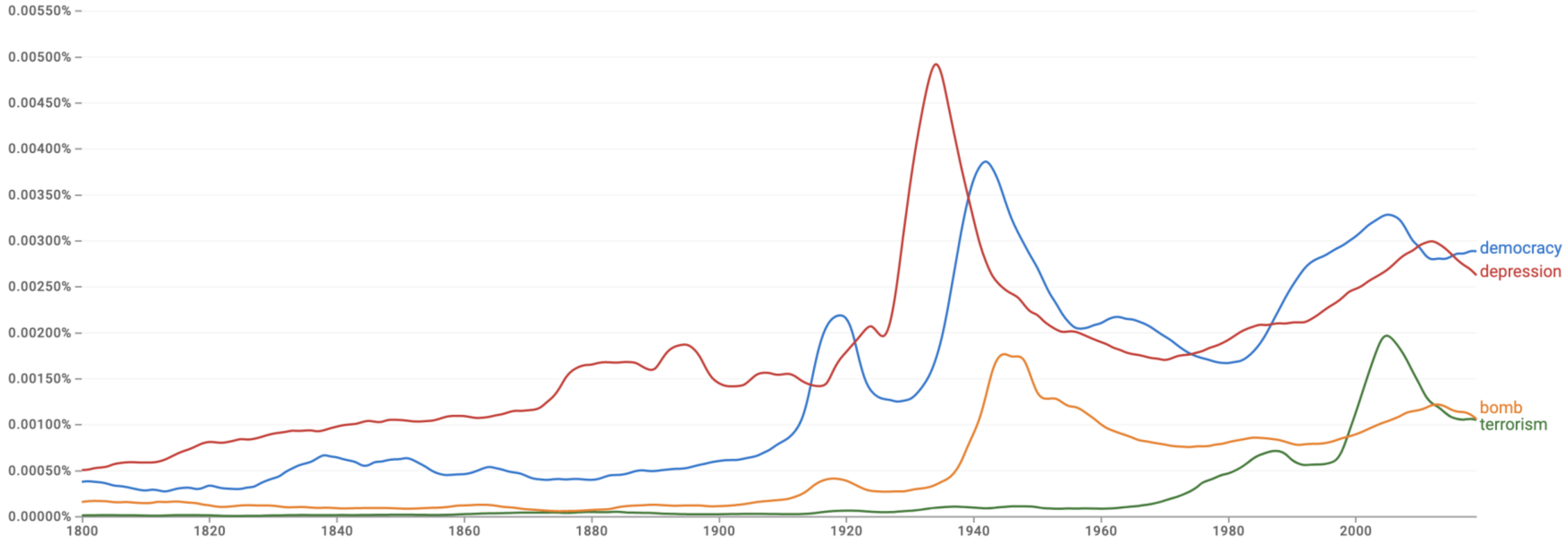
N-Grams LMs and Long-range Dependencies

In general, count-based LMs are insufficient models of language because language has long-distance dependencies:

"The computer which I had just put into
the machine room on the fifth floor crashed."

Pre-Computed N-Grams

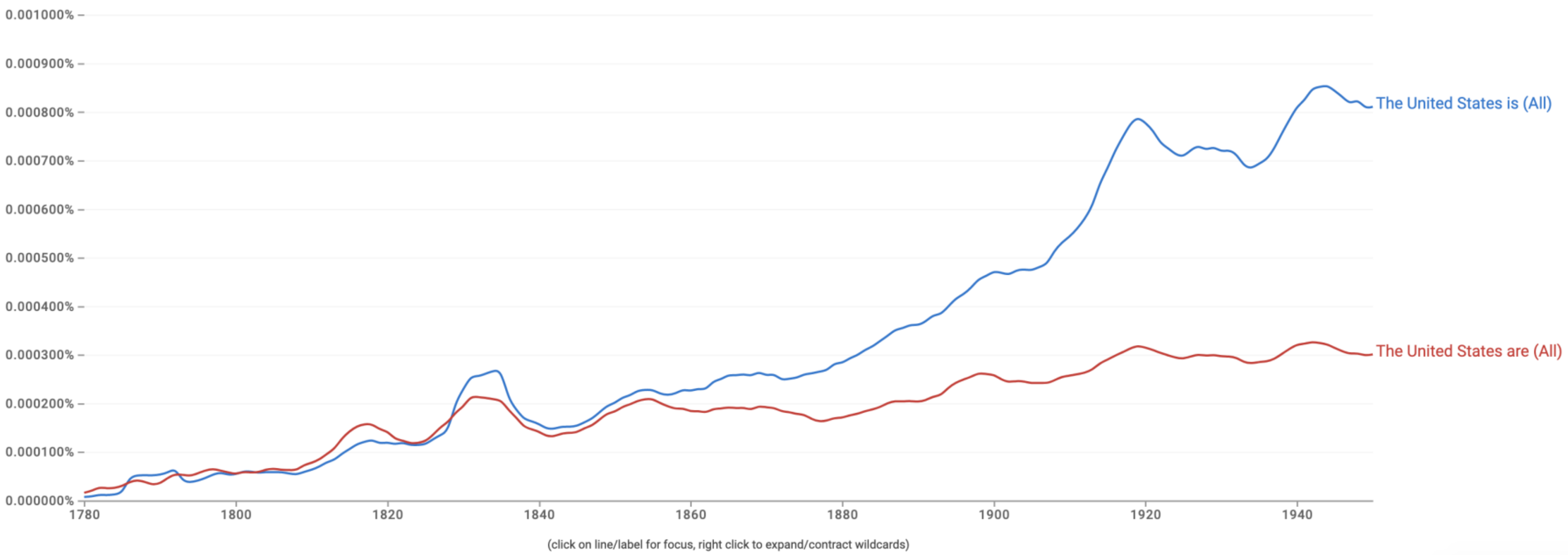
Google Books Ngram Viewer



- Google n-gram viewer https://books.google.com/ngrams/
- Data: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Pre-Computed N-Grams

Google Books Ngram Viewer

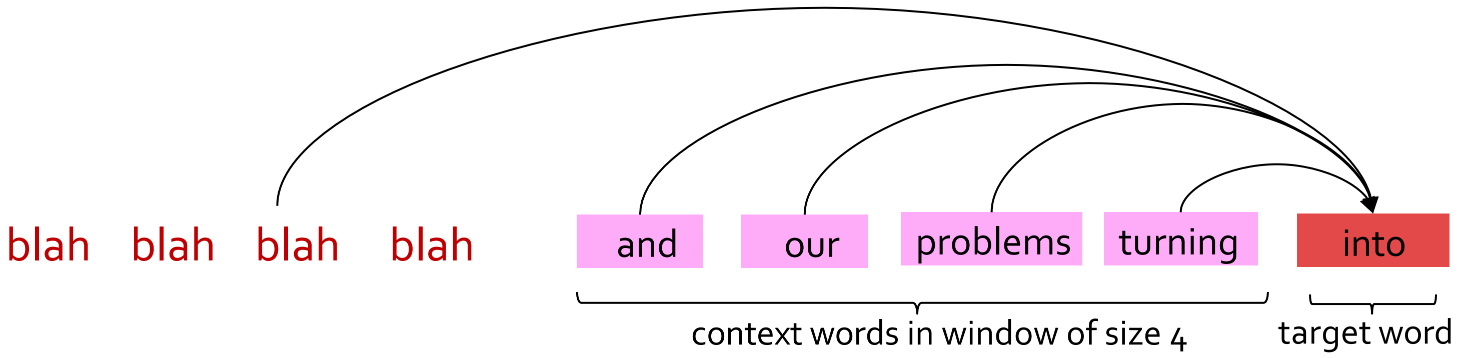


- Google n-gram viewer https://books.google.com/ngrams/
- Data: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

LM as a Machine Learning Problem

Task: Given the embeddings of the context, predict the word on the right side.

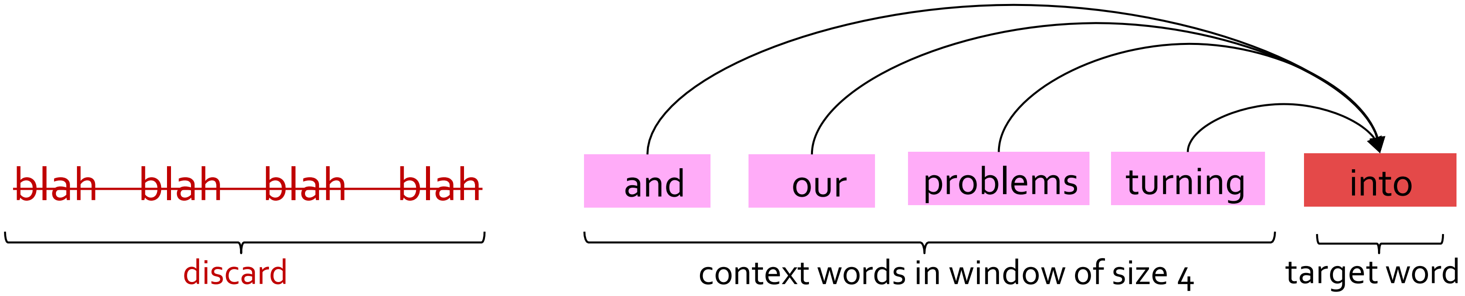
Discard anything beyond its context window



LM as a Machine Learning Problem

Task: Given the embeddings of the context, predict the word on the right side.

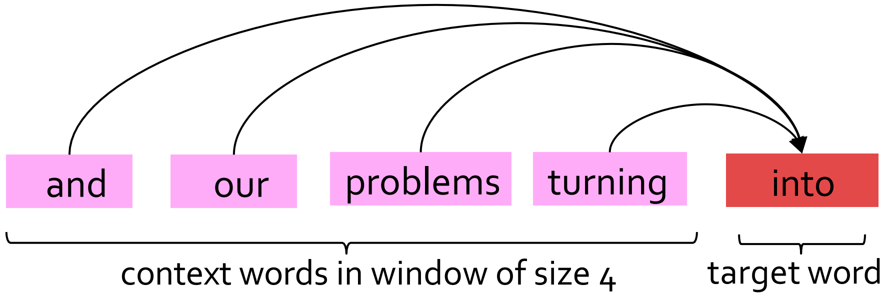
Discard anything beyond its context window



LM as a Machine Learning Problem

Task: Given the embeddings of the context, predict the word on the right side.

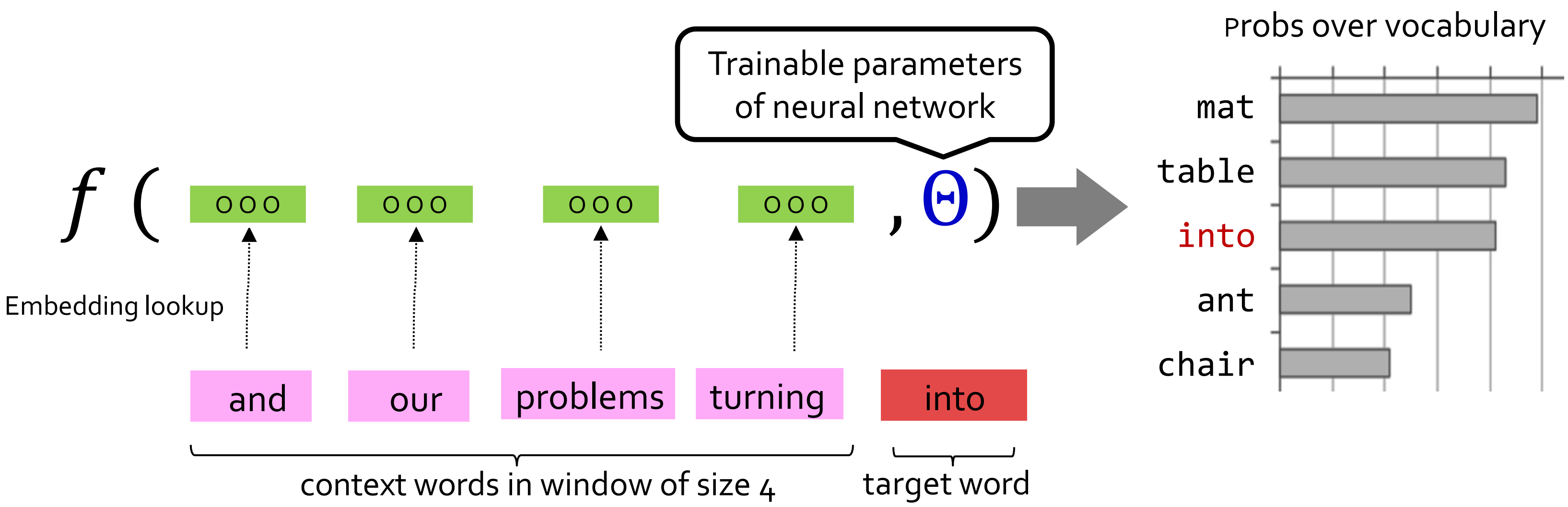
Discard anything beyond its context window



A Fixed-Window Neural LM

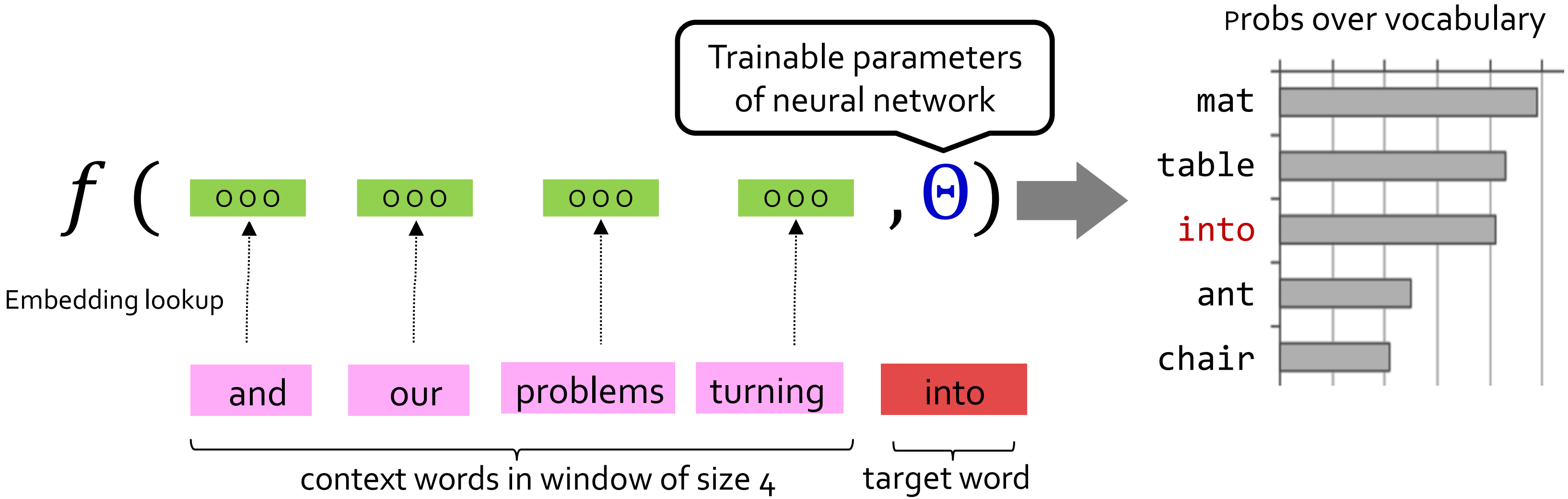
TECHNISCHE UNIVERSITÄT DARMSTADT

Training this model is basically optimizing its parameters Θ such that it assigns high probability to the target word.



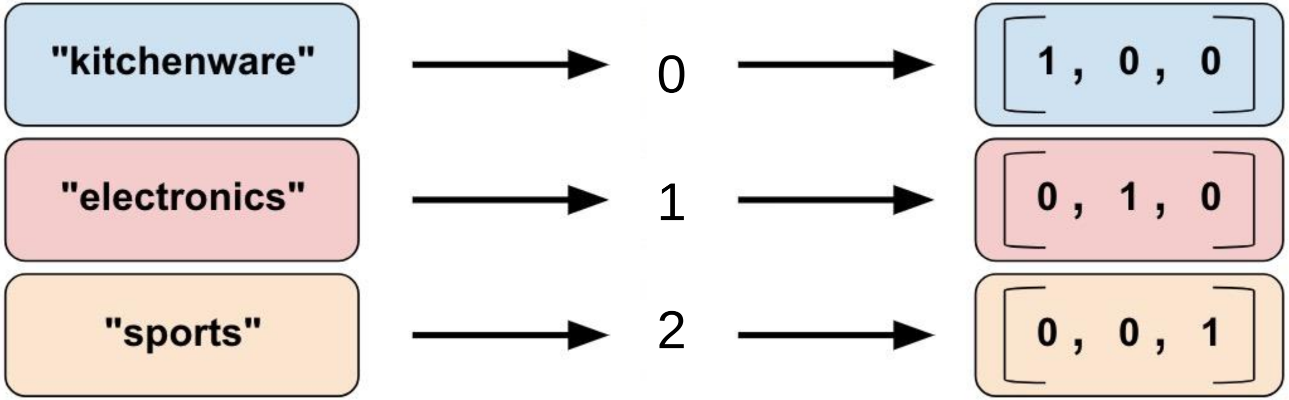
A Fixed-Window Neural LM

- It will also lay the foundation for the future models (RNN, transformers, ...)
- But first we need to figure out how to train neural networks!



Feeding Text to Neural Net

- In practice this is implemented in this way:
  1. Turn each word into a unique index
  2. Map each index into a one-hot vector



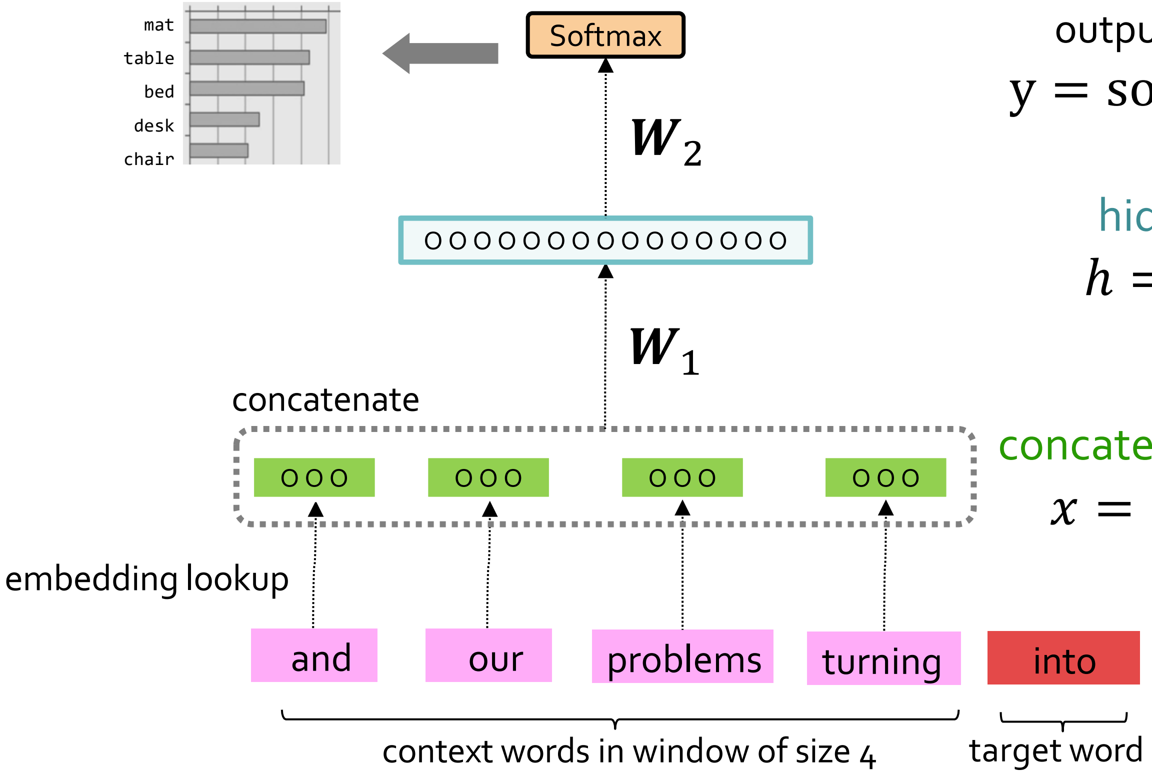
Feeding Text to Neural Net

TECHNISCHE UNIVERSITÄT DARMSTADT

- In practice this is implemented in this way:
  1. Turn each word into a unique index
  2. Map each index into a one-hot vector
  3. Lookup the corresponding word embedding via matrix multiplication

$$\underbrace{\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix}}_{\text{One-hot vector}} \times \underbrace{\begin{bmatrix}8 & 2 & 1 & 9\\ 6 & 5 & 4 & 0\\ 7 & 1 & 6 & 2\\ 1 & 3 & 5 & 8\\ 0 & 4 & 9 & 1\end{bmatrix}}_{\text{Embedding Weight Matrix}} = \underbrace{\begin{bmatrix}1 & 3 & 5 & 8\end{bmatrix}}_{\text{Hidden layer output}}$$

A Fixed-Window Neural LM



output distribution

$$y = \text{softmax}(W_2 h)$$

hidden layer

$$h = f(W_1 x)$$

concatenated word embeddings

$$x = [v_1, v_2, v_3, v_4]$$

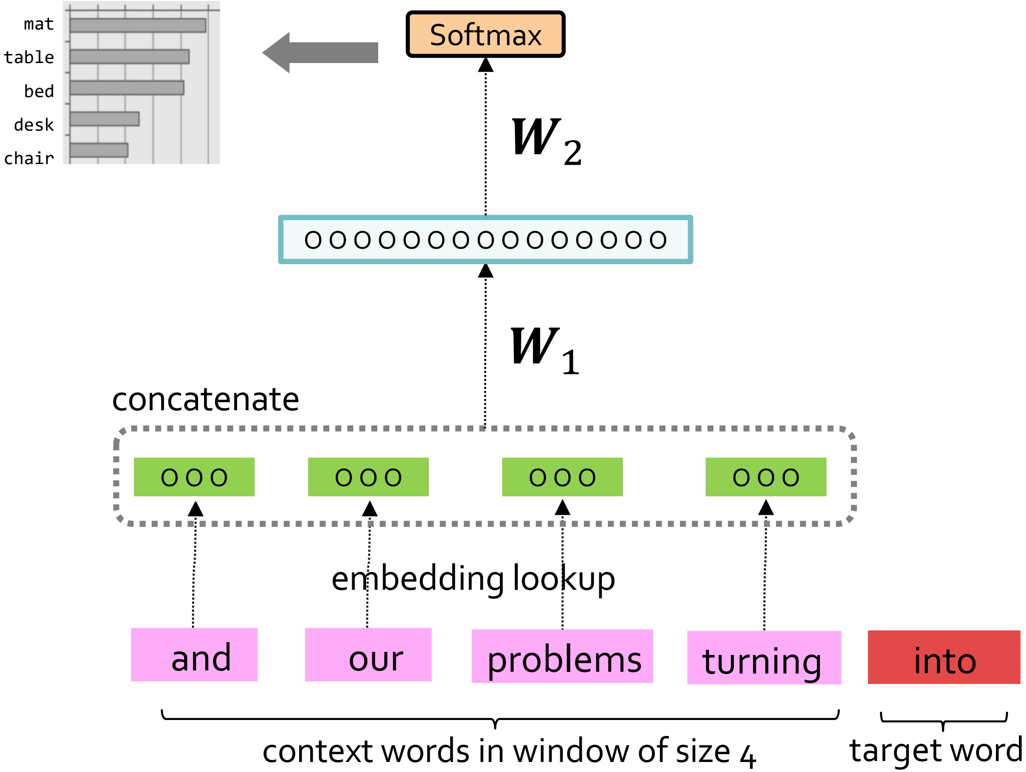
A Fixed-Window Neural LM

Improvements over n-gram LM:

- Tackles the sparsity problem
- Model size is O(n) not O(exp(n)) — n being the window size.

Remaining problems:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$ — Window can never be large enough!
- It’s not deep enough to capture nuanced contextual meanings



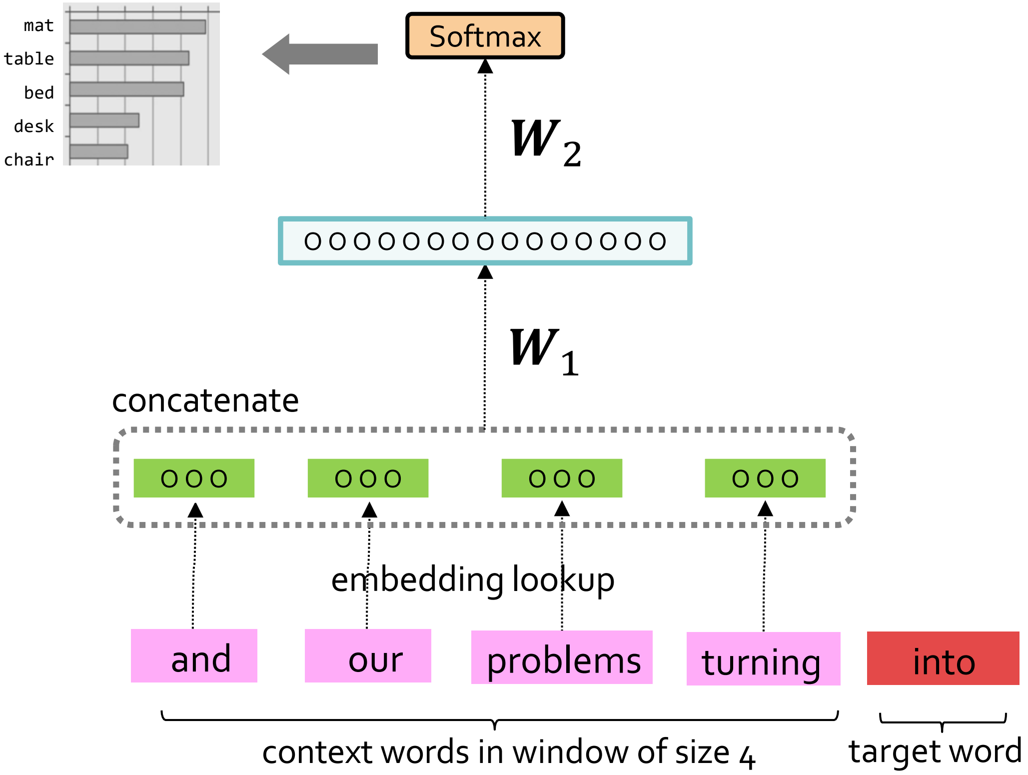
A Fixed-Window Neural LM

Improvements over n-gram LM:

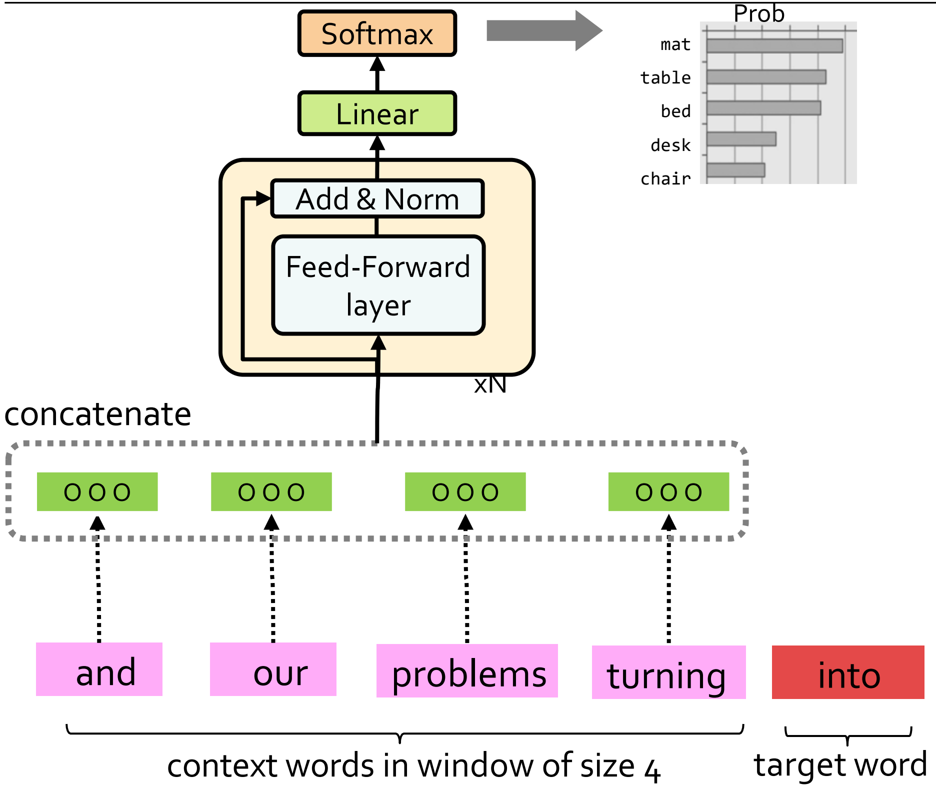
- Tackles the sparsity problem
- Model size is O(n) not O(exp(n)) — n being the window size.

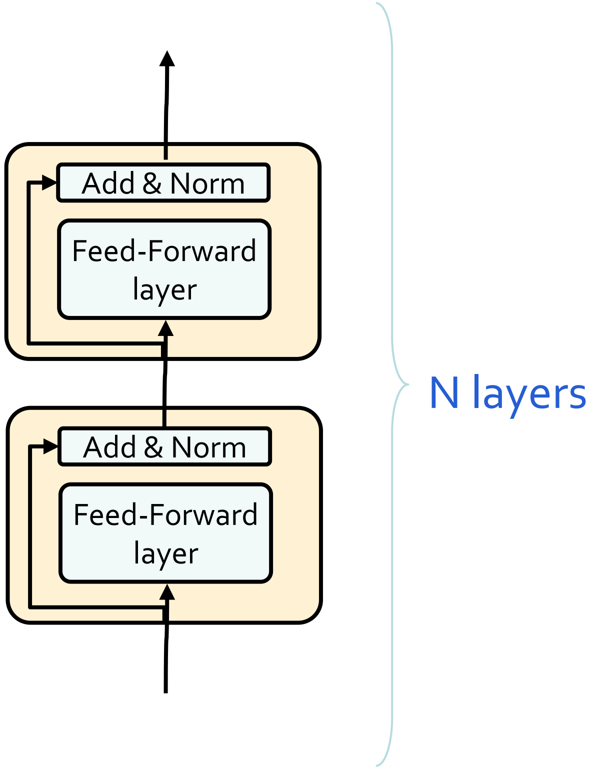
Remaining problems:

- Fixed window is too small
- Enlarging window enlarges ***W*** — Window can never be large enough!
- It's not deep enough to capture nuanced contextual meanings

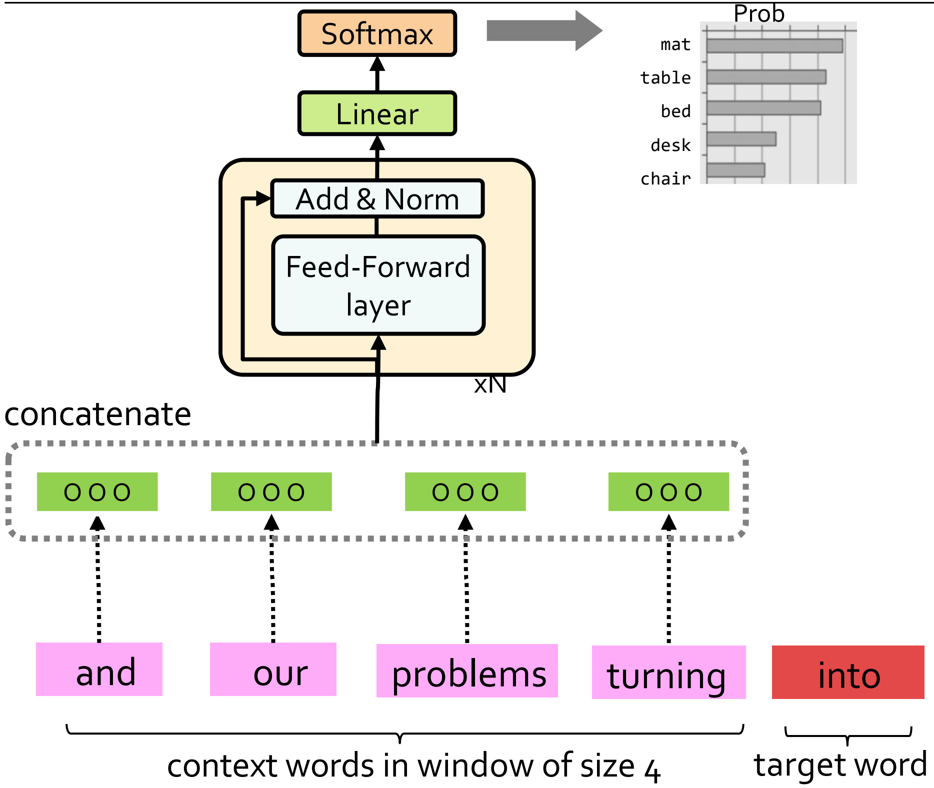


Sun and Iyyer (2021): Revisiting Simple Neural Probabilistic Language Models





Sun and Iyyer (2021): Revisiting Simple Neural Probabilistic Language Models



Model	# Params	Val. perplexity
Transformer	148M	25.0
NPLM-old	32M[2]	216.0
NPLM-old (large)	221M[3]	128.2
NPLM 1L	123M	52.8
NPLM 4L	128M	38.3
NPLM 16L	148M	**31.7**
- Residual connections	148M	660.0
- Adam, + SGD	148M	418.5
- Layer normalization	148M	33.0

Table 1: NPLM model ablation on WIKITEXT-103.

What Changed from N-Gram LMs to Neural LMs?

TECHNISCHE UNIVERSITÄT DARMSTADT

What is the source of Neural LM's strength?

Why sparsity is less of an issue for Neural LMs?

Answer: In n-grams, we treat all prefixes independently of each other! (even those that are semantically similar)

```
students opened their ___
pupils opened their ___
scholars opened their ___
undergraduates opened their ___
students turned the pages of their ___
students attentively perused their ___
...
```

Neural LMs are able to share information across these semantically-similar prefixes and overcome the sparsity issue.

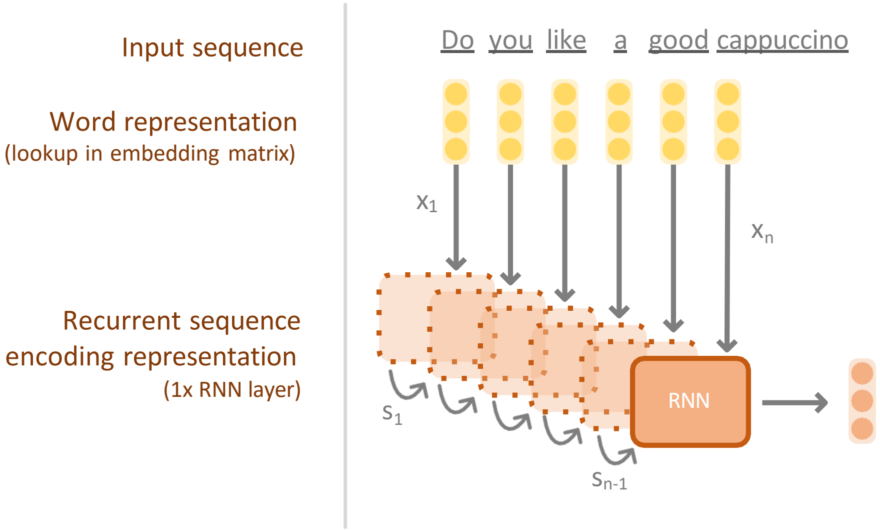
Outline

Recap: LM and Sub-word tokenization

Basic Neural Language Models

RNN and Transformer LM

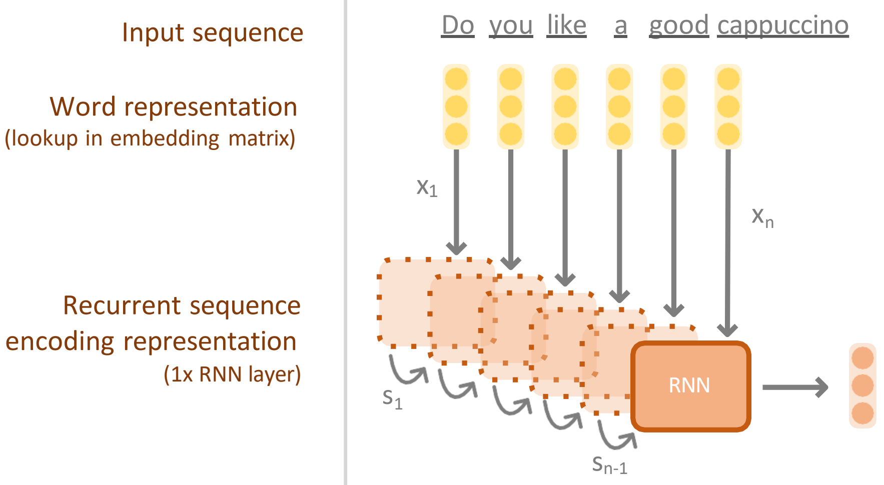
RNN Language Model

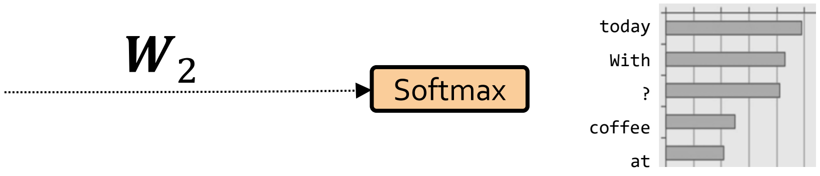


- Sequence as the input
- Single vector (last state) as output
  - Part of a larger network

RNN Language Model

- Sequence as the input
- Single vector (last state) as output
  - Part of a larger network





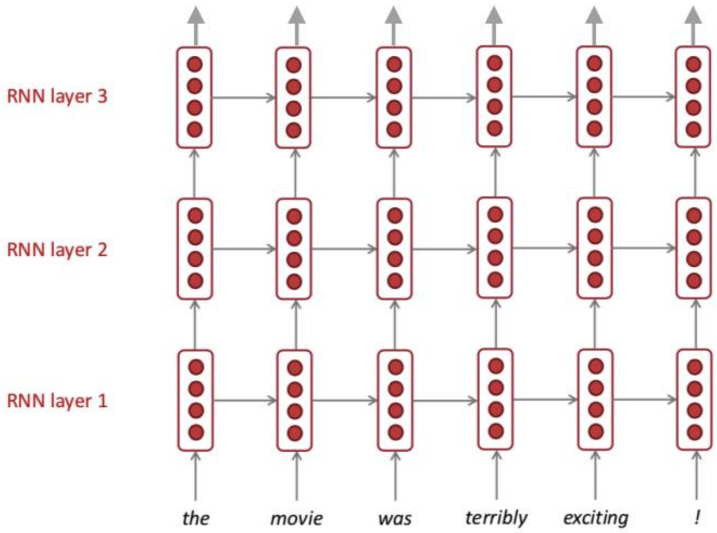
RNN Language Model

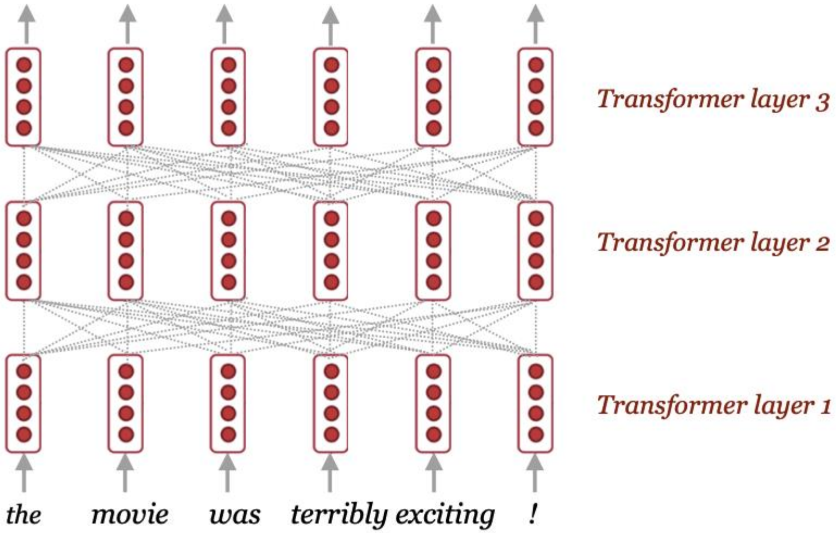
What are the cons?

- While RNNs in theory can represent long sequences, they quickly forget portions of the input.
- Vanishing/exploding gradients
- Difficult to parallelize

What can we do?

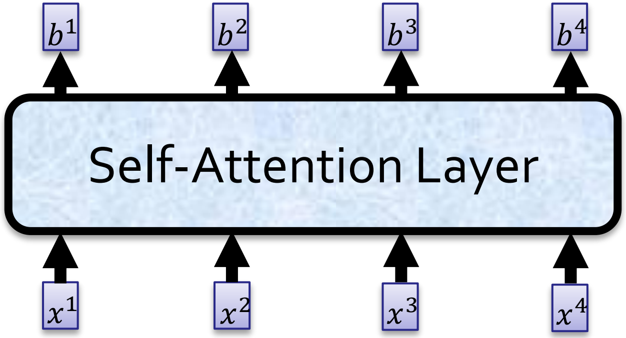
RNN vs Transformer





Self-Attention: Back to Big Picture

- Attention is a powerful mechanism to create context-aware representations
- A way to focus on select parts of the input



- Better at maintaining long-distance dependencies in the context.

[Attention Is All You Need, Vaswani et al. 2017]

Properties of Self-Attention

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$

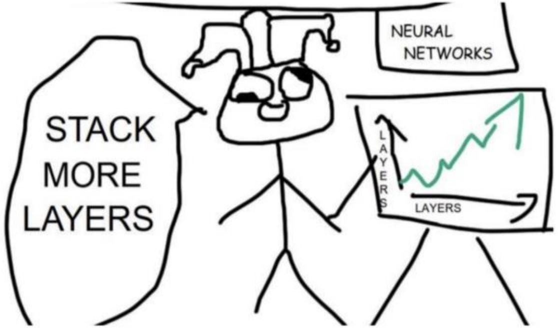
- n = sequence length, d = hidden dimension
- Quadratic complexity, but:
  - O(1) sequential operations (not linear like in RNN)

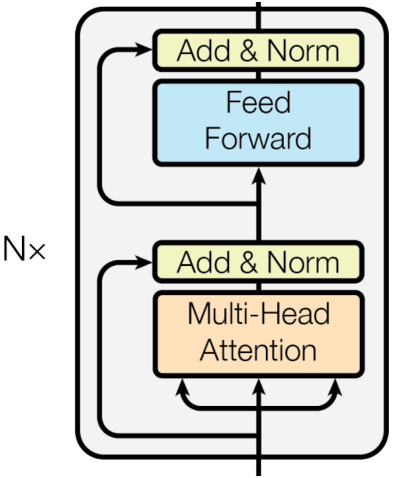
- Efficient implementations

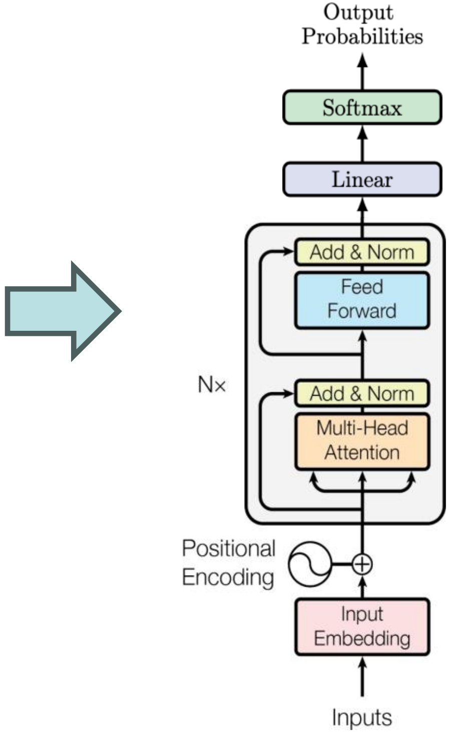
[Attention Is All You Need, Vaswani et al. 2017]

How Do We Make it Deep?

- Step 1: Stack more layers!
- Step 2: …
- Step 3: Profit!







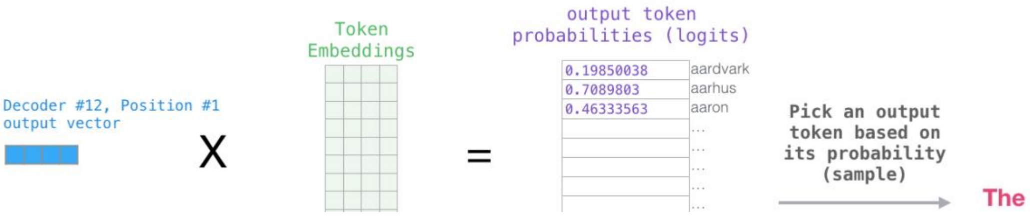
From Representations to Prediction

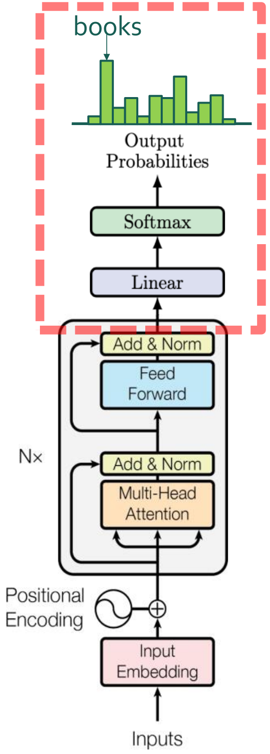
- To perform prediction, add a classification head on top of the final layer of the transformer.
- This can be per token (Language modeling)
- Or can be for the entire sequence (only one token)

$$\text{out} \in \mathbb{R}^{S \times d} \quad \text{(S: Sequence length)}$$

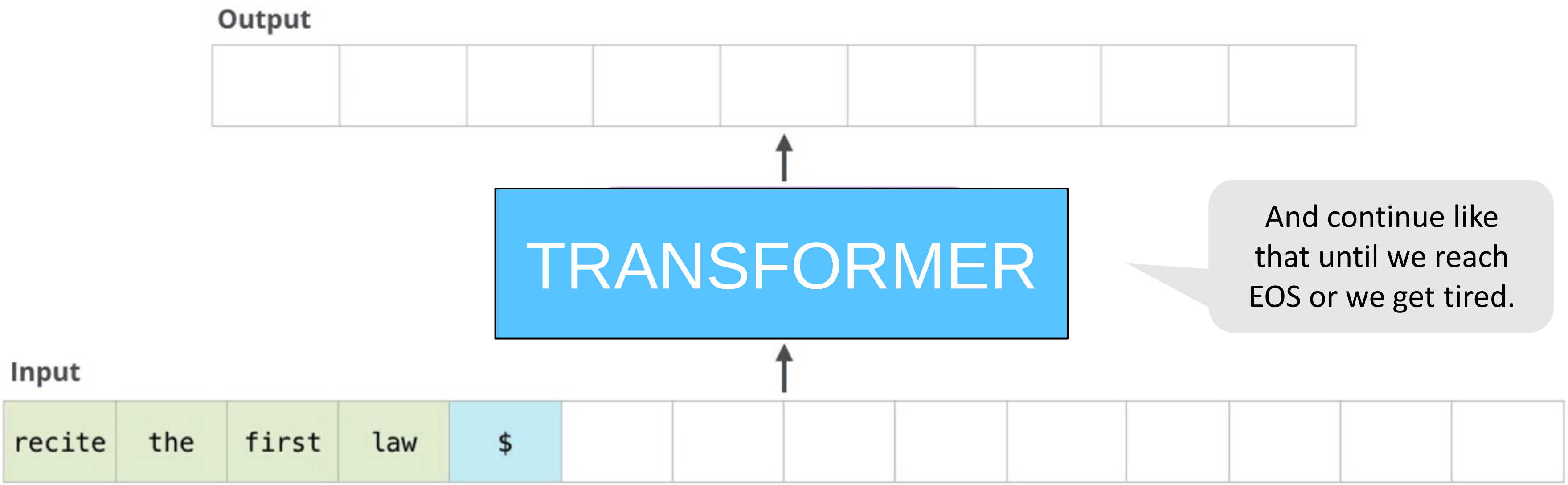
$$\text{logits} = \text{Linear}_{(\text{d, V})}(out) = f\left(out \,.\, W_V\right) \in \mathbb{R}^{S \times V}$$

$$\text{probabilies} = \text{softmax}(\text{logits}) \in \mathbb{R}^{S \times V}$$





Transformer-based Language Modeling

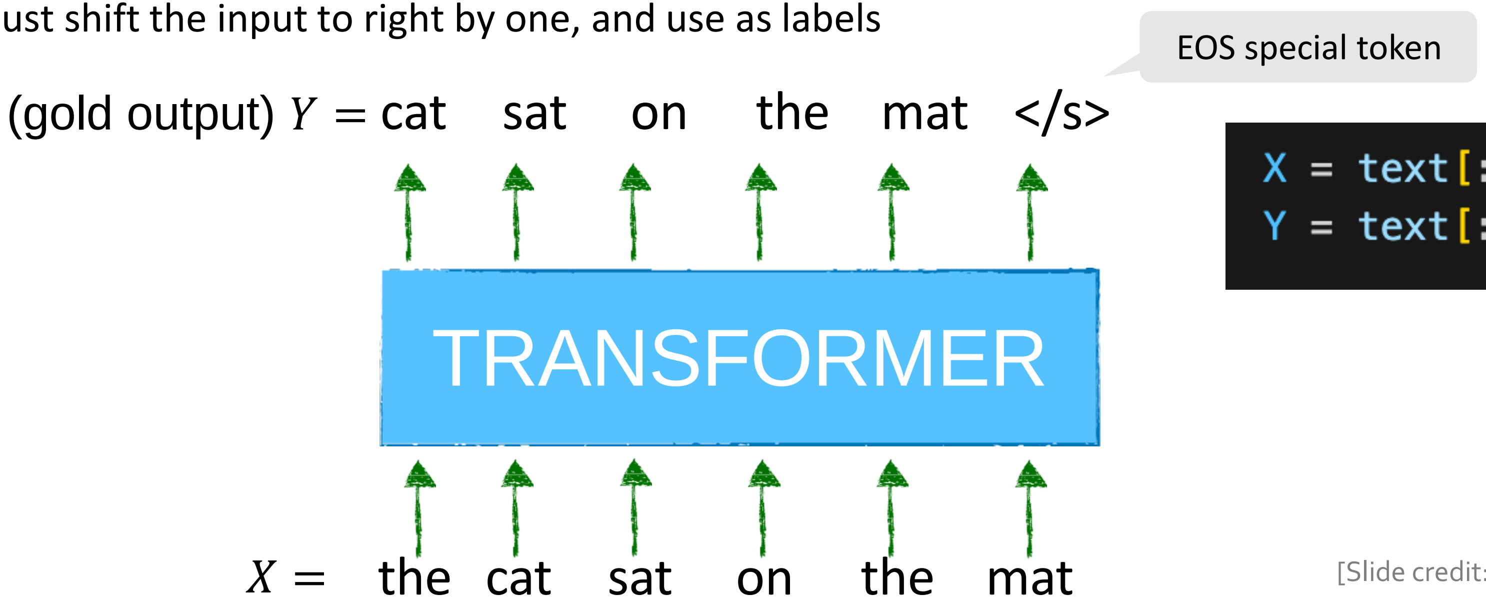


And continue like that until we reach EOS or we get tired.

Image by http://jalammar.github.io/illustrated-gpt2/

Training a Transformer Language Model

- **Goal:** Train a Transformer for language modeling (i.e., predicting the next word).
- **Approach:** Train it so that each position is predictor of the next (right) token.
  - We just shift the input to right by one, and use as labels



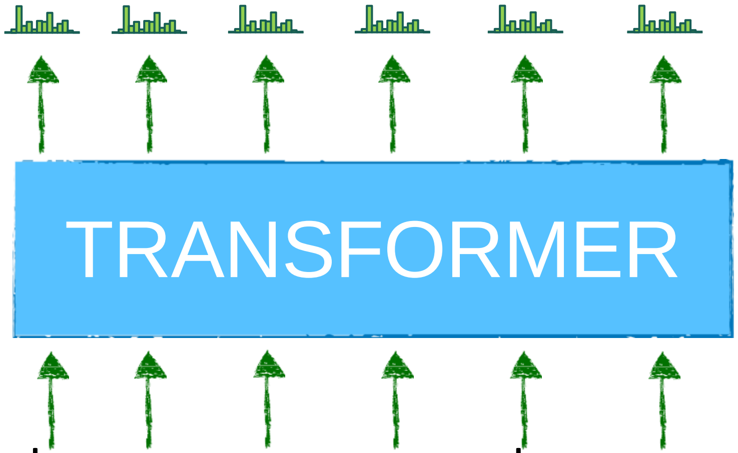

```
X = text[:, :-1]
Y = text[:, 1:]
```

[Slide credit: Arman Cohan]

Training a Transformer Language Model

- For each position, compute their corresponding **distribution** over the whole vocab.

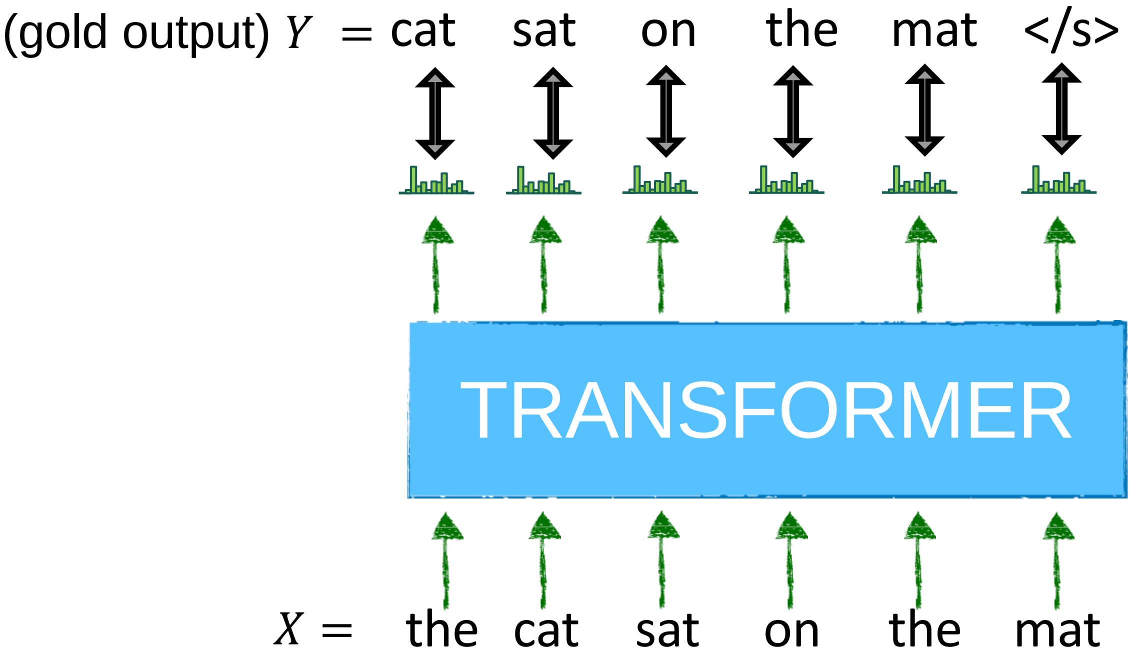
(gold output) Y = cat sat on the mat </s>



X = the cat sat on the mat

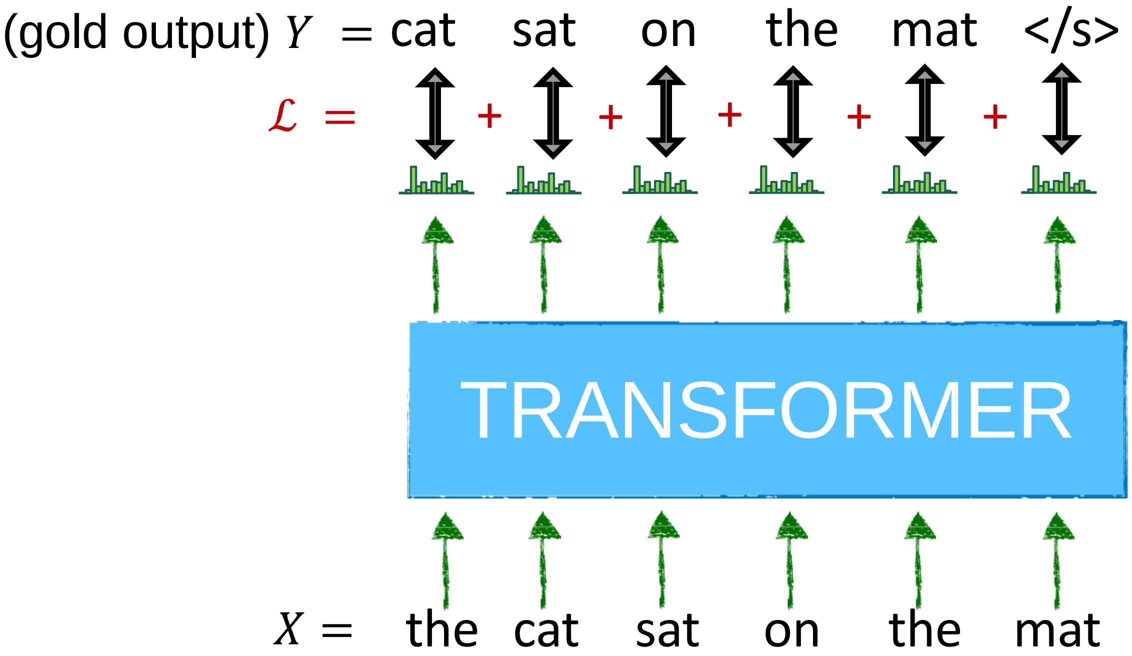
Training a Transformer Language Model

- For each position, compute the **loss** between the distribution and the gold output label.



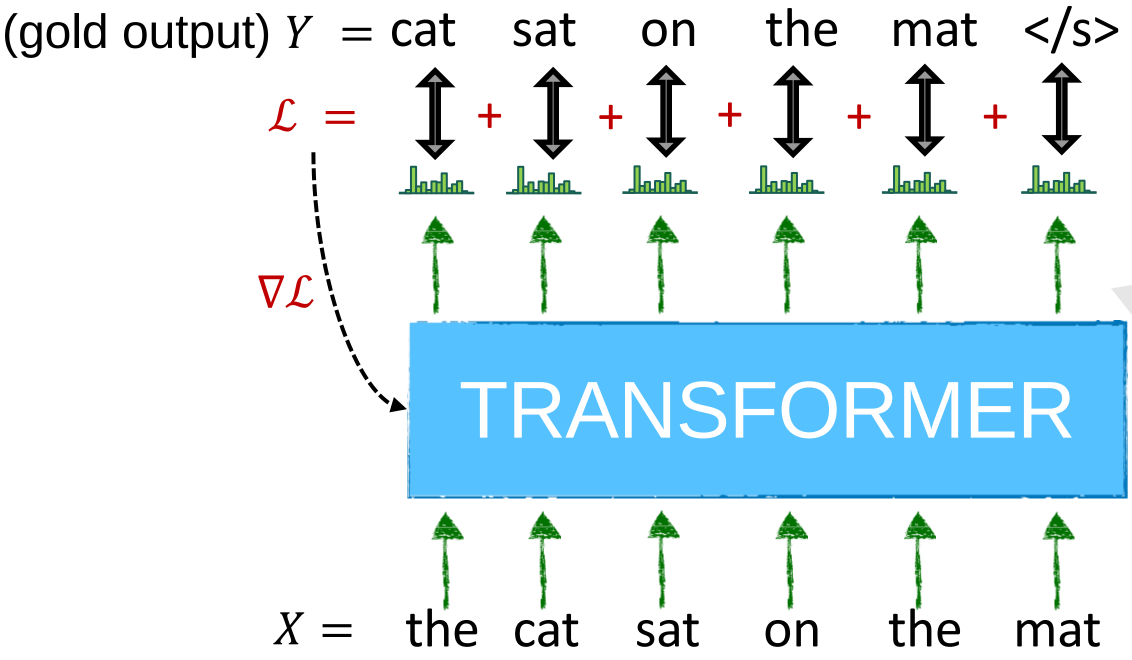
Training a Transformer Language Model

- Sum the position-wise loss values to a obtain a **global loss**.



Training a Transformer Language Model

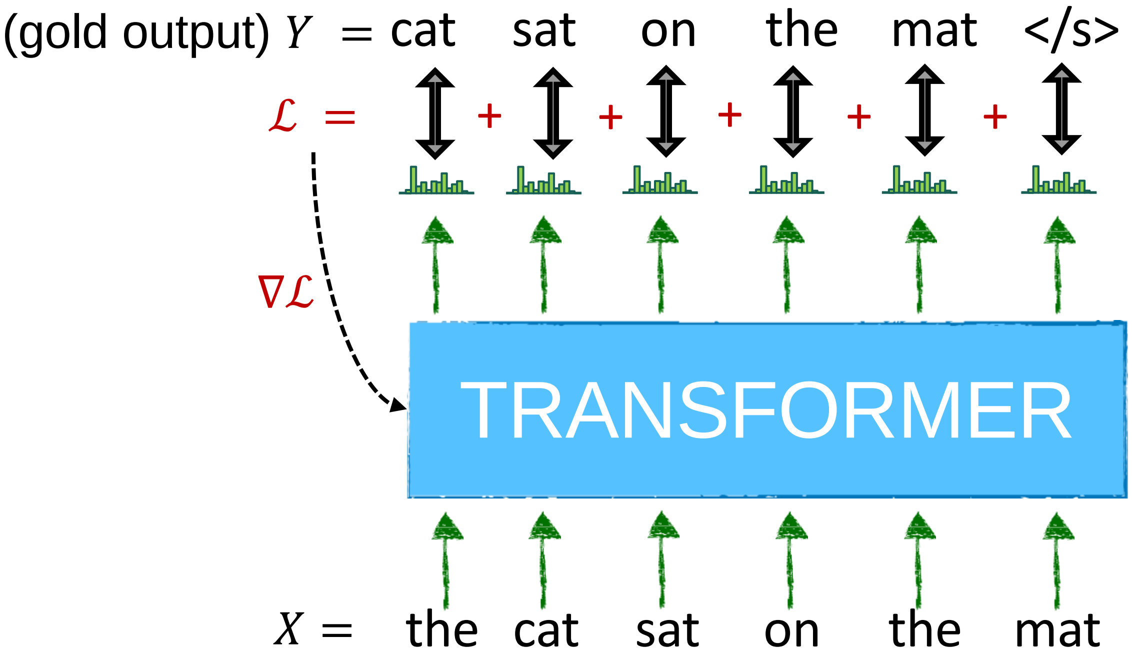
- Using this loss, do **Backprop** and **update** the Transformer parameters.



Well, this is not quite right … what is the problem with this?

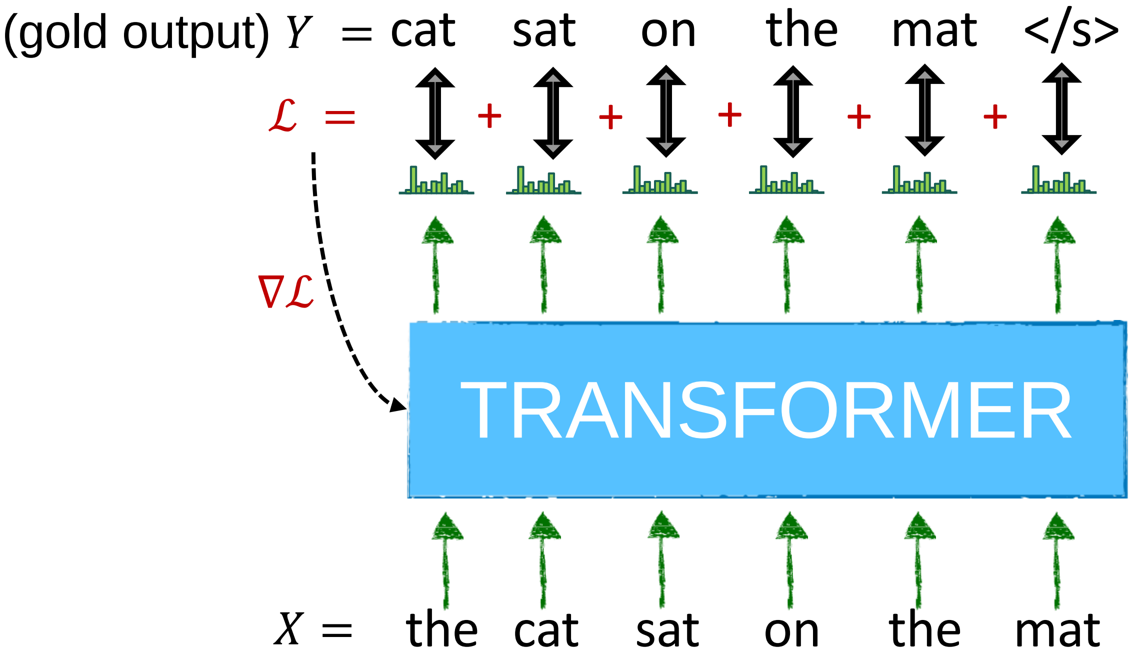
Training a Transformer Language Model

- The model would solve the task by copying the next token to output (data leakage).

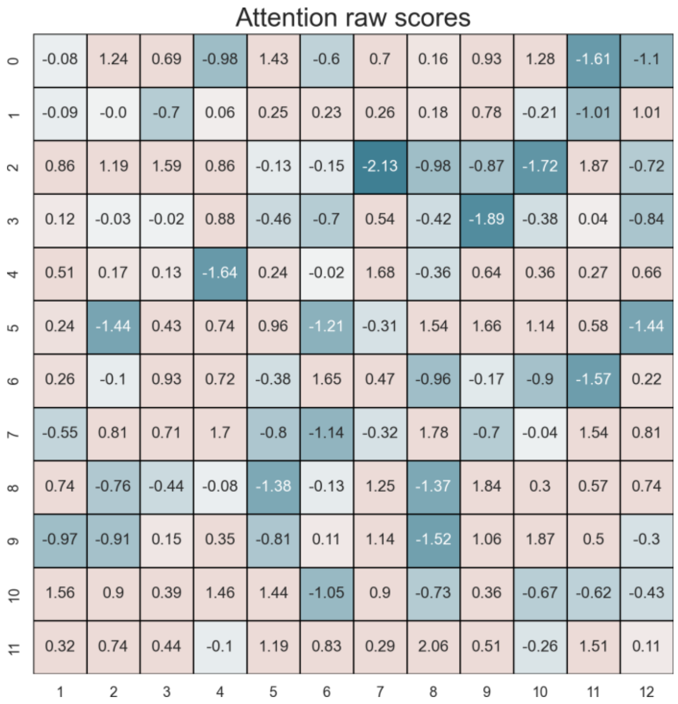


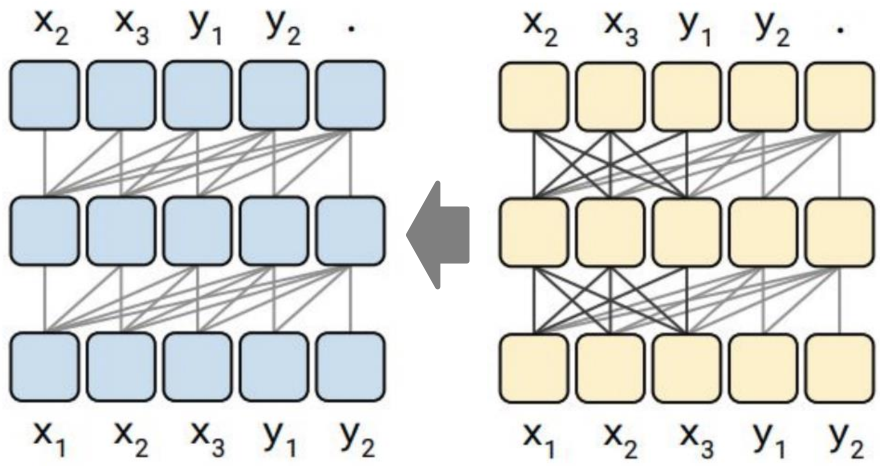
Training a Transformer Language Model

- We need to **prevent information leakage** from future tokens! How?



Attention mask





What we want

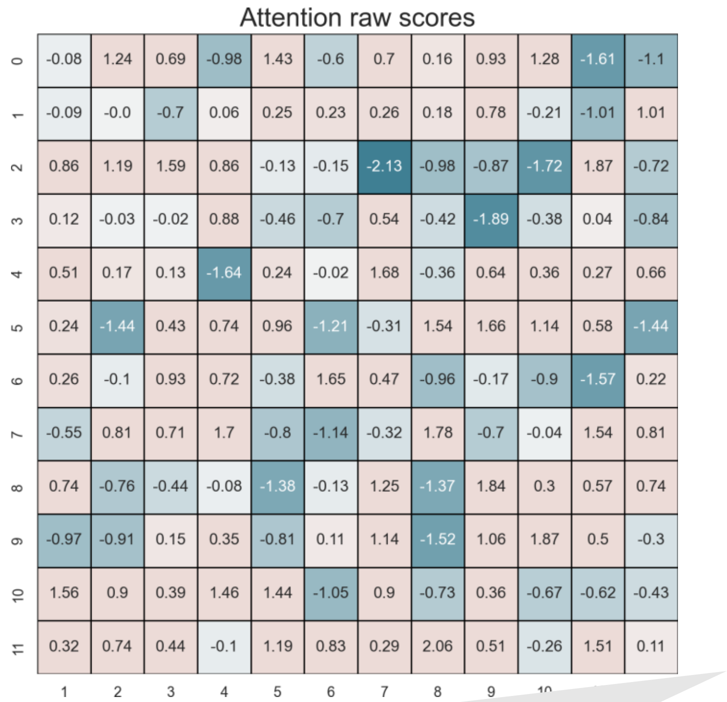
What we have

Attention mask

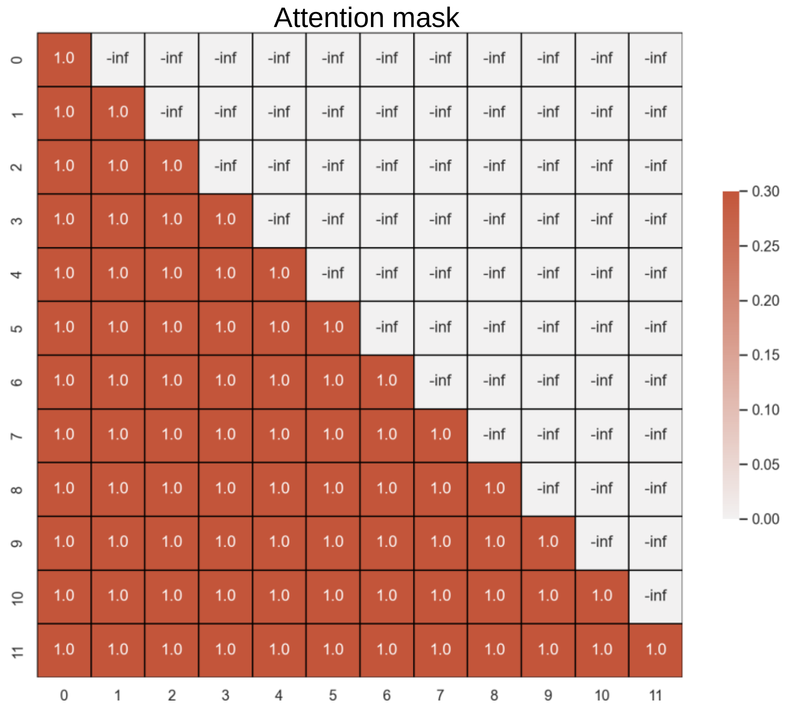
Attention raw scores

Attention mask

Attention mask

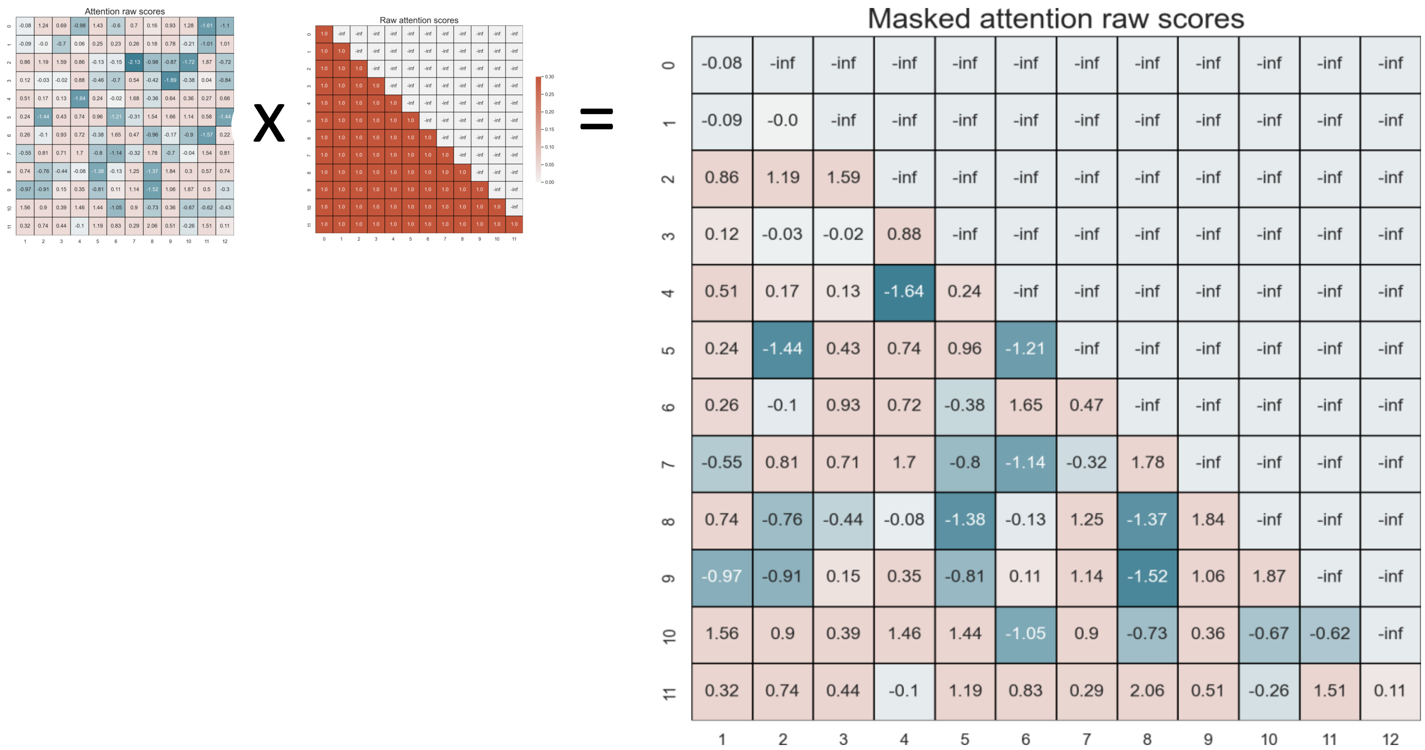


X

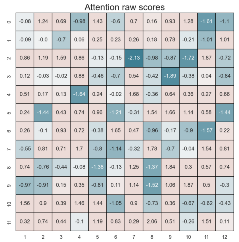


Note matrix multiplication is quite fast in GPUs.

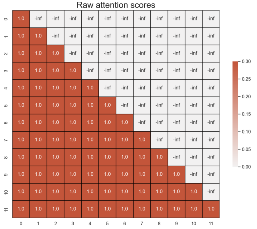
Attention mask

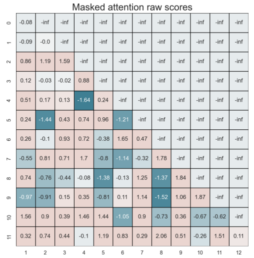


Attention mask

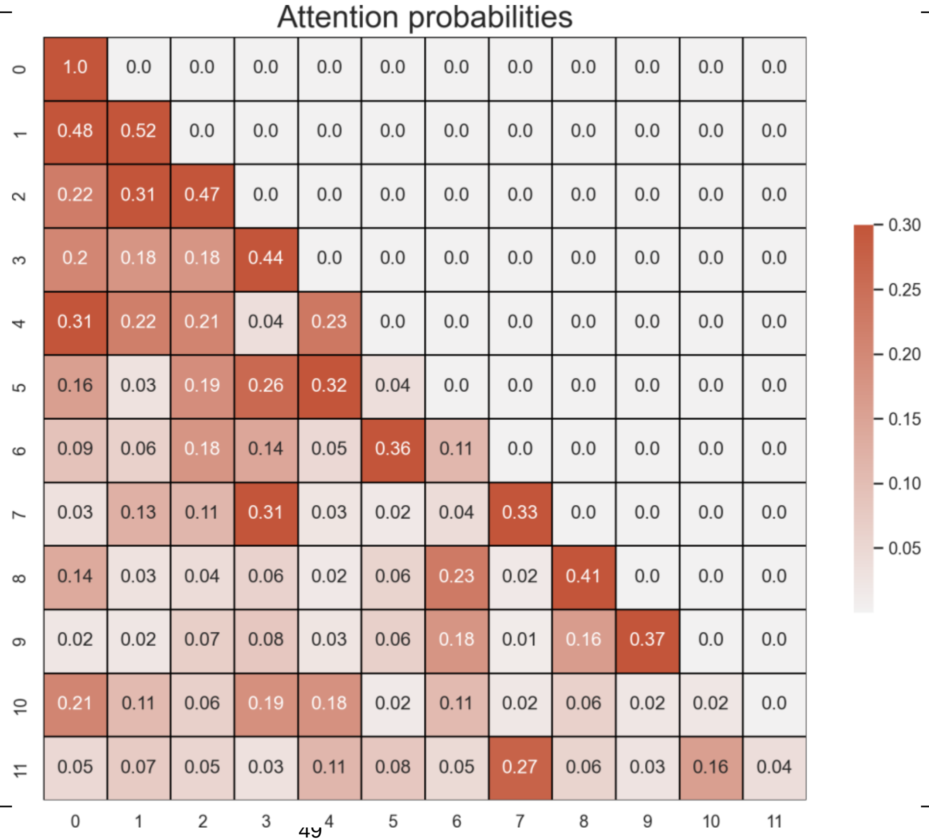


X



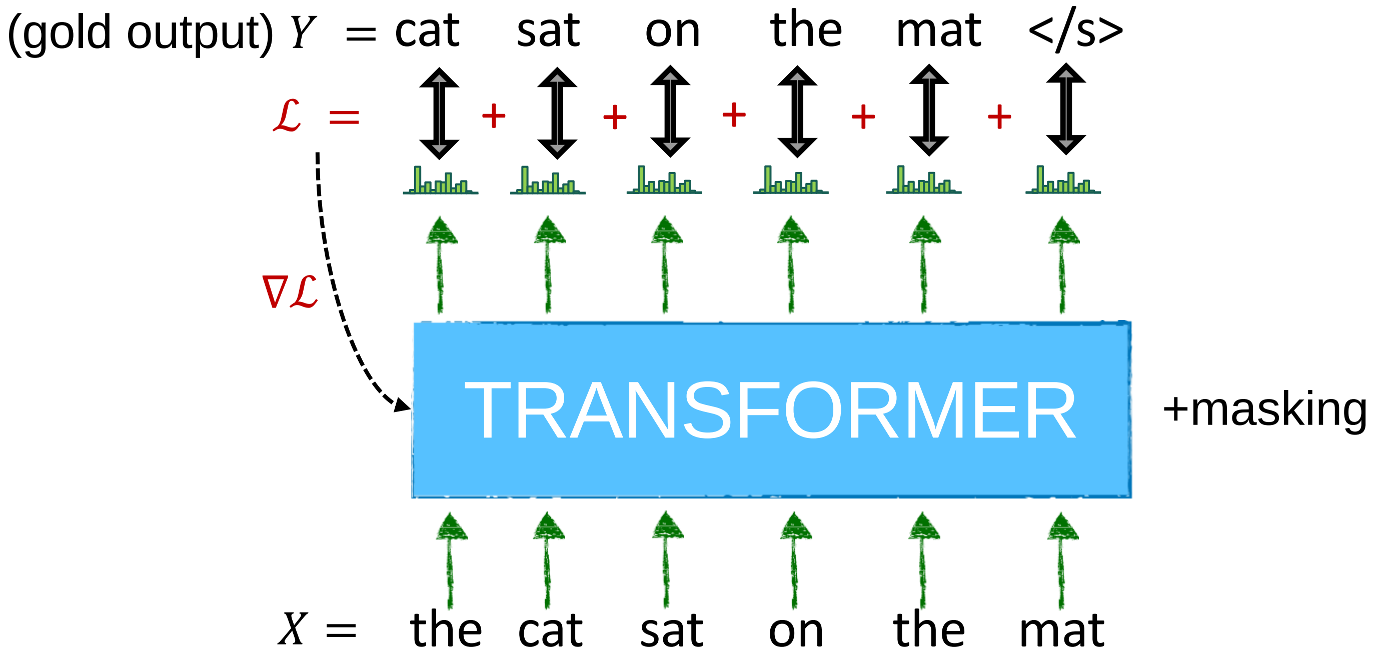


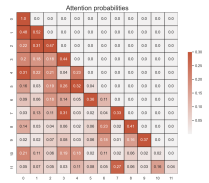
softmax



Training a Transformer Language Model

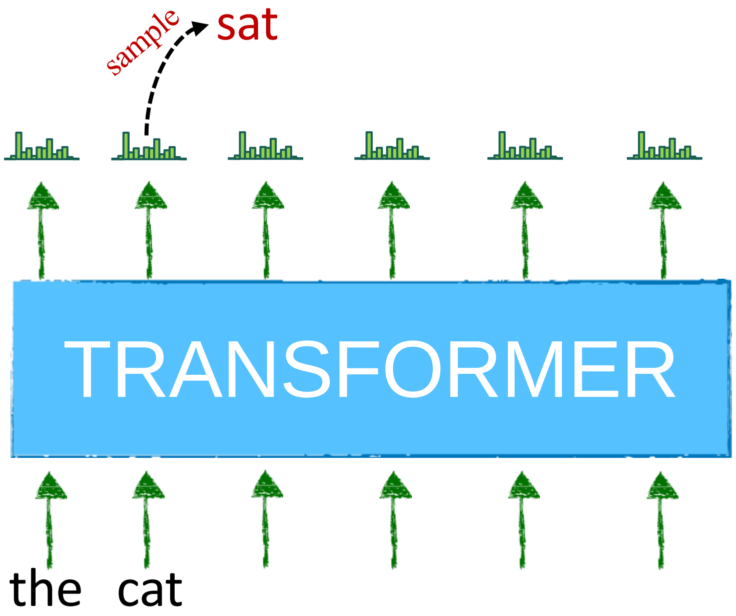
- We need to **prevent information leakage** from future tokens! How?





How to use the model to generate text?

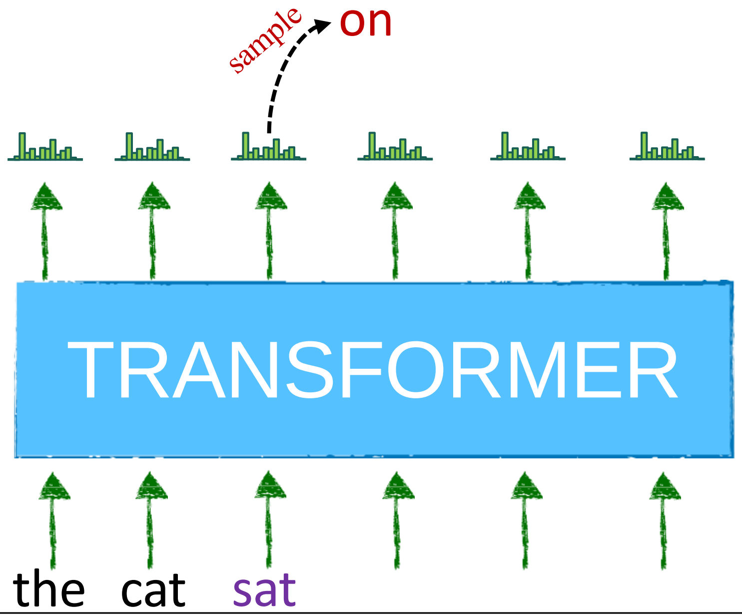
- Use the output of previous step as input to the next step repeatedly



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

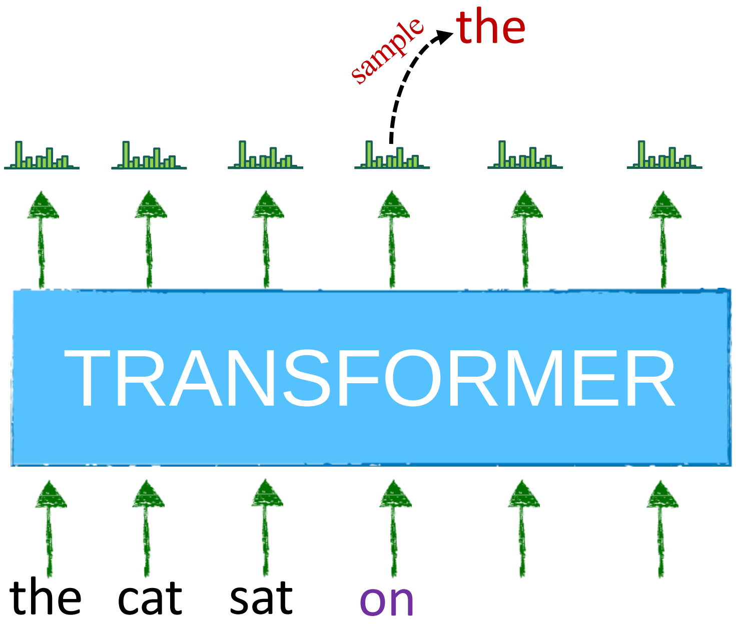
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

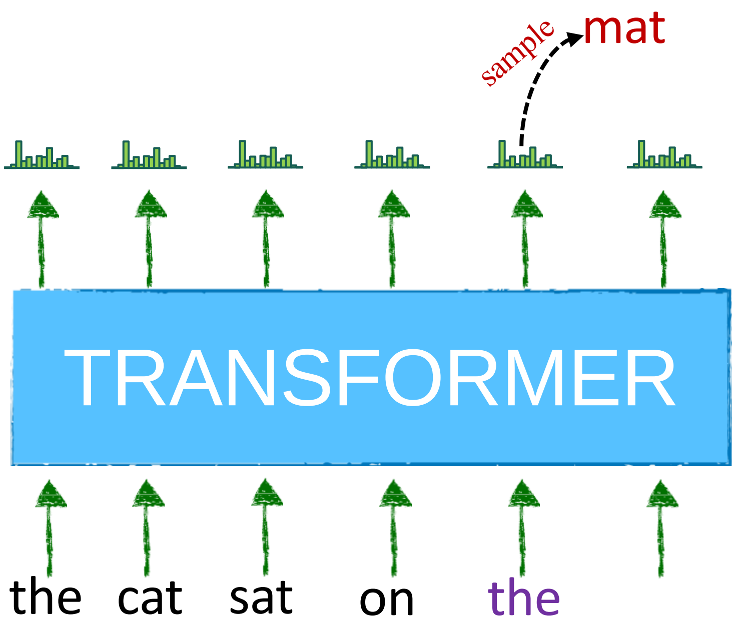
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

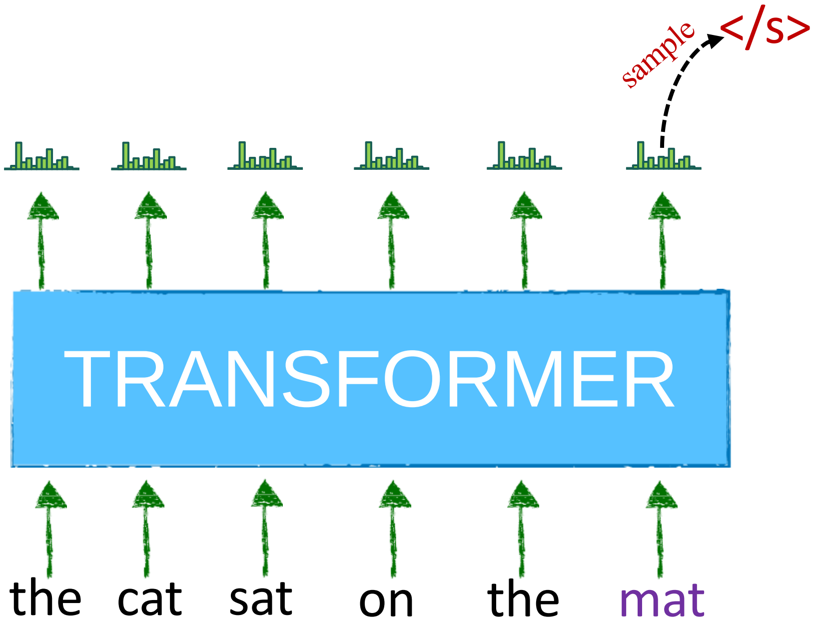
The probabilities get revised upon adding a new token to the input.



How to use the model to generate text?

- Use the output of previous step as input to the next step repeatedly

The probabilities get revised upon adding a new token to the input.



Lessons Learned

- Neural models overcome n-gram limitations (sparsity, fixed windows) and model long-range dependencies.
- Fixed-window models reduce sparsity but lack depth and scalability for larger contexts.
- RNNs handle sequences but struggle with vanishing gradients and parallelization.
- Transformers use self-attention for efficient, context-aware representations, excelling in long-distance dependencies.
- Masking prevents future token leakage during training, enabling autoregressive text generation.

Next Lecture

Adaptation & Prompting