Course 3: Language Modeling

You

Explain how ChatGPT works as if I was a 5 years old child.

ChatGPT

Alright, imagine you have a really smart friend who knows a lot of things. That friend is like a big, magical library with lots and lots of books. Now, this friend is called ChatGPT. When you ask ChatGPT a question or tell it something, it looks into all those books in its library (which are actually a huge amount of information it learned) and tries to give you the best answer or response it can find.

So, you can think of ChatGPT as a helpful buddy who loves to chat and share what it knows with you!

How does it really work?

What is Language Modeling?

Definition

- A sequence of tokens $(w_1, w_2, \ldots, w_n)$
- For a position i, a language model (**LM**) predicts

$$P(w_i \,|(w_j)_{j \neq i}) \in \Delta^V$$

- In words: a LM predicts the probability of a token given its context

Example

I went to the ***???*** *yesterday*

P(*park* | *I went to the* ***???*** *yesterday*) = 0.1

P(*zoo* | *I went to the* ***???*** *yesterday*) = 0.07

…

P(*under* | *I went to the* ***???*** *yesterday*) = 0

Why is it hard?

- **Large vocabularies**: 170,000 English words
- **Lots of possible contexts**:
  - For V possible tokens, there are V^L contexts of size L (in theory)
- **Inherent uncertainty**: not obvious even for humans

Basic approach - Unigram

- Learn the *non-contextual* probability (=frequency) of each token:

$$P(w_i \,|(w_j)_{j \neq i}) = f$$

Example

chart against operations at influence the surface plays crown a inaro the three @ but the court lewis on hand american of seamen mu role due roger executives

Include context - Bigram

- Predict based on the last token only:

$$P(w_i \mid (w_j)_{j \neq i}) = P_\theta(w_i | w_{i-1})$$

- (MLE): Measure next token frequency

Example

the antiquamen lost to dios nominated former is carved stone oak were problematic, 1910. his willingness to receive this may have been seen anything

Include more context - n-gram

- Predict based on the n last tokens only:

$$P(w_i \,|(w_j)_{j\neq i}) = P_\theta(w_i | w_{i-n} \ldots w_{i-1})$$

- (MLE): Measure occurences of tokens after $w_{i-n} \ldots w_{i-1}$

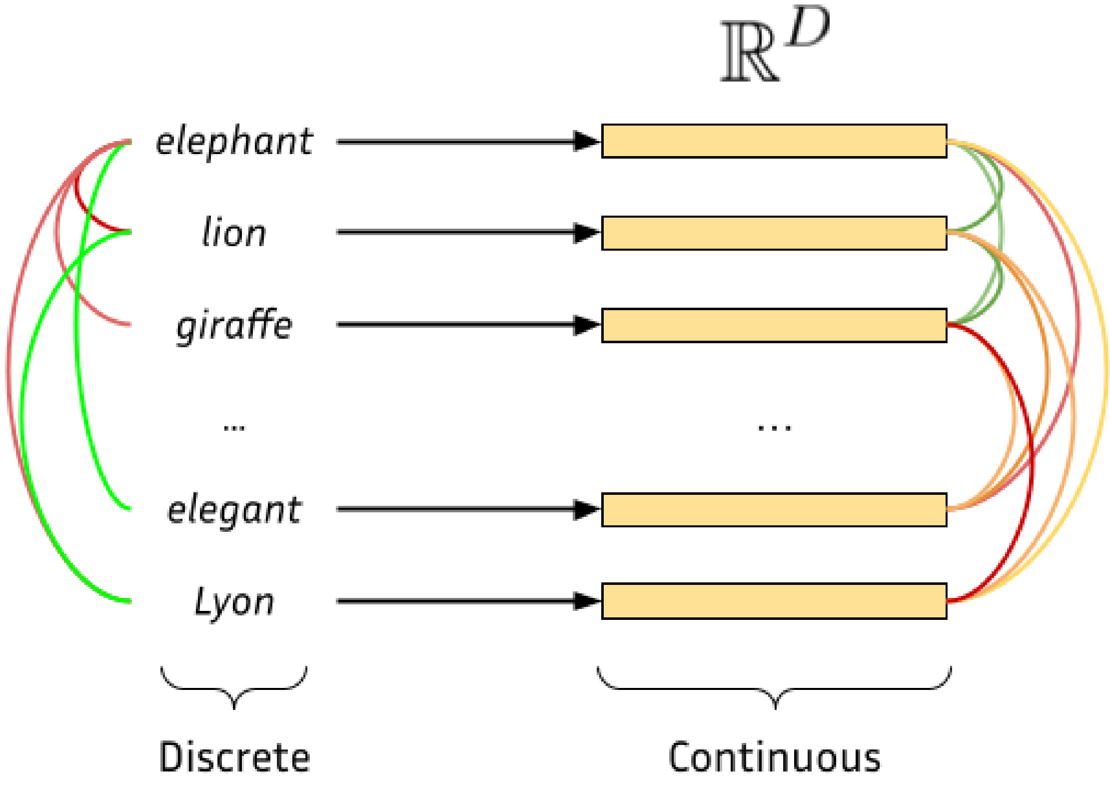
Example (n=4)

eva gauthier performed large amounts of contemporary french music across the united states marshals service traveled to frankfurt, germany and took custody of the matthews

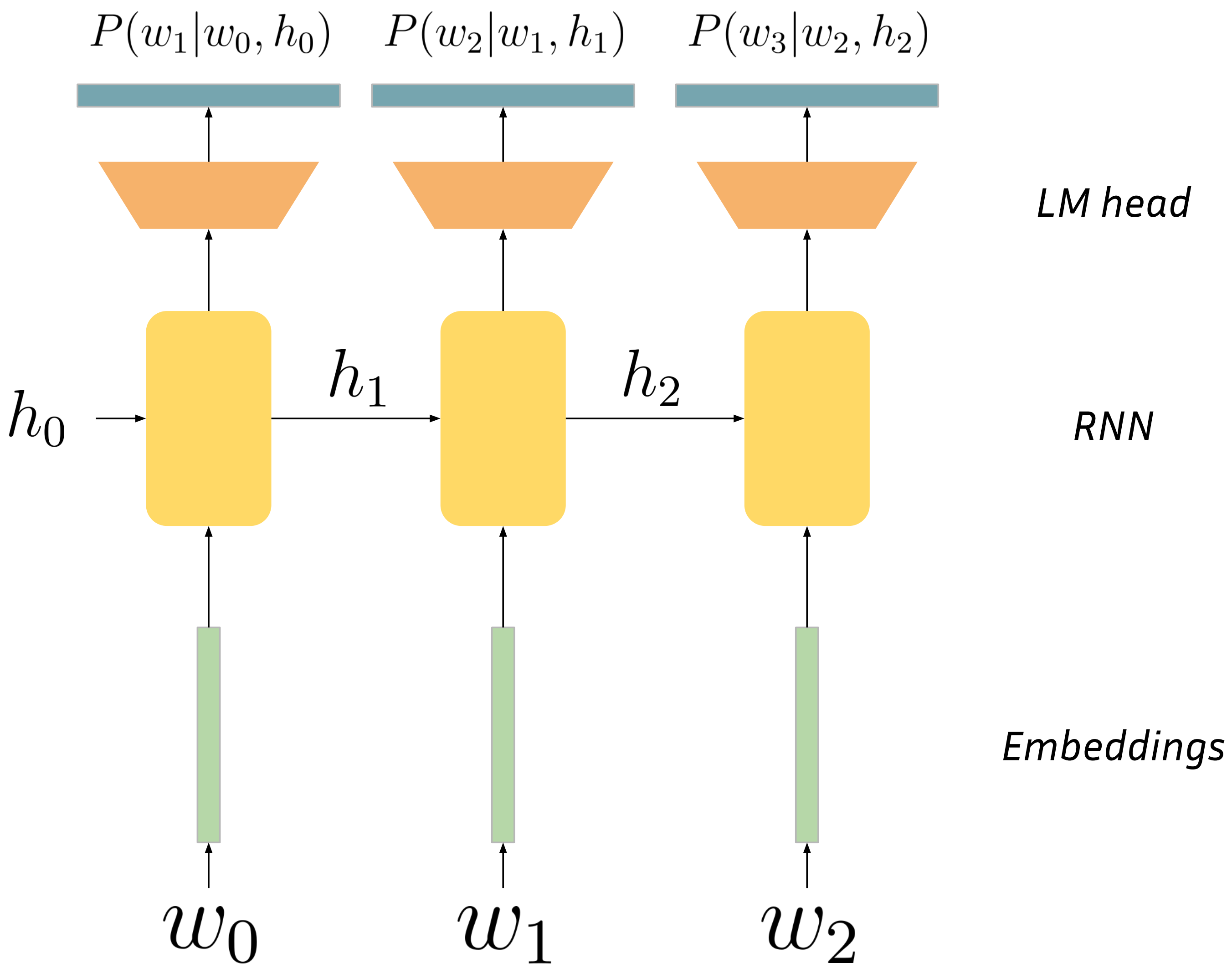
Statistical n-grams: pro/cons

- Strenghts:
  - Easy to train
  - Easy to interpret
  - Fast inference
- Limitations:
  - Very limited context
  - **Unable to extrapolate** : can only model what it has seen

The embedding paradigm



LM with RNNs


$P(w_1|w_0, h_0)$
$P(w_2|w_1, h_1)$
$P(w_3|w_2, h_2)$
LM head
h_0
h_1
h_2
RNN
Embeddings
w_0
w_1
w_2

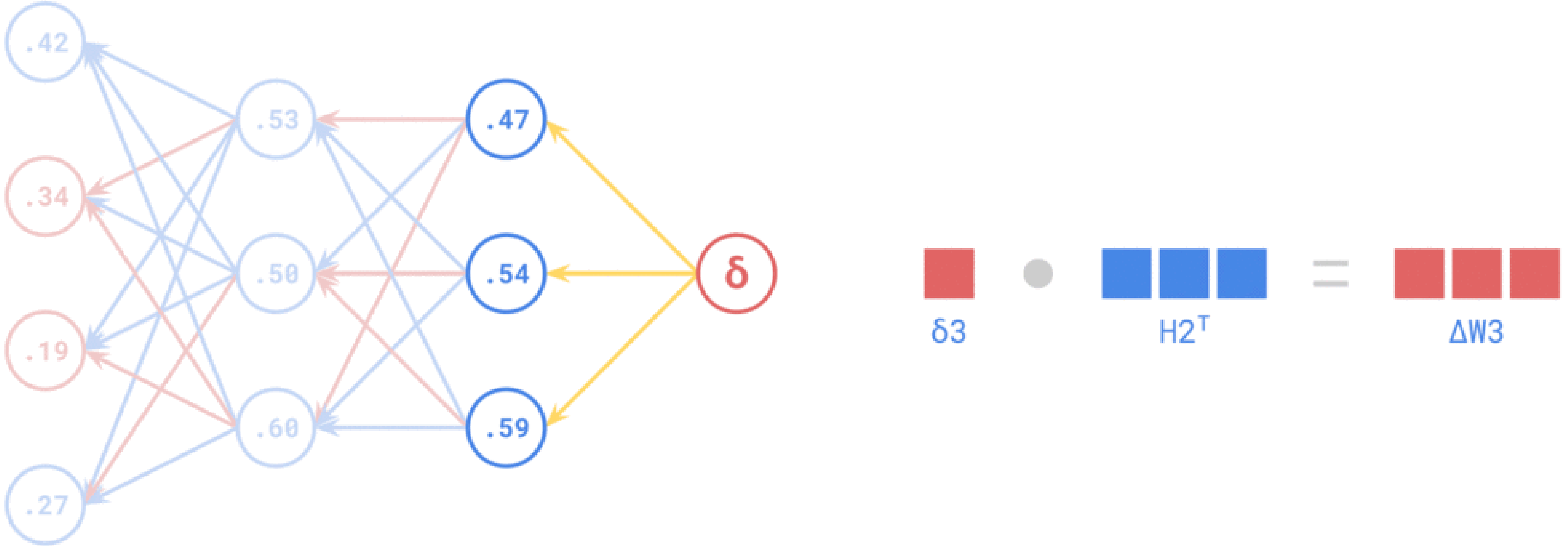
LM with RNNs - Training

- θ: parameters of the RNN
- $(w_1, \ldots, w_n)$: training sequence
- Cross-entropy loss $\mathcal{L}_{ce}$:

$$\mathcal{L}_{ce}(w, \theta) = -\sum_{i=2}^{n} 1_{w_i} \cdot \log P_\theta(w_i | w_{i-1}, h_{i-1})$$

- Train via back-propagation + SGD

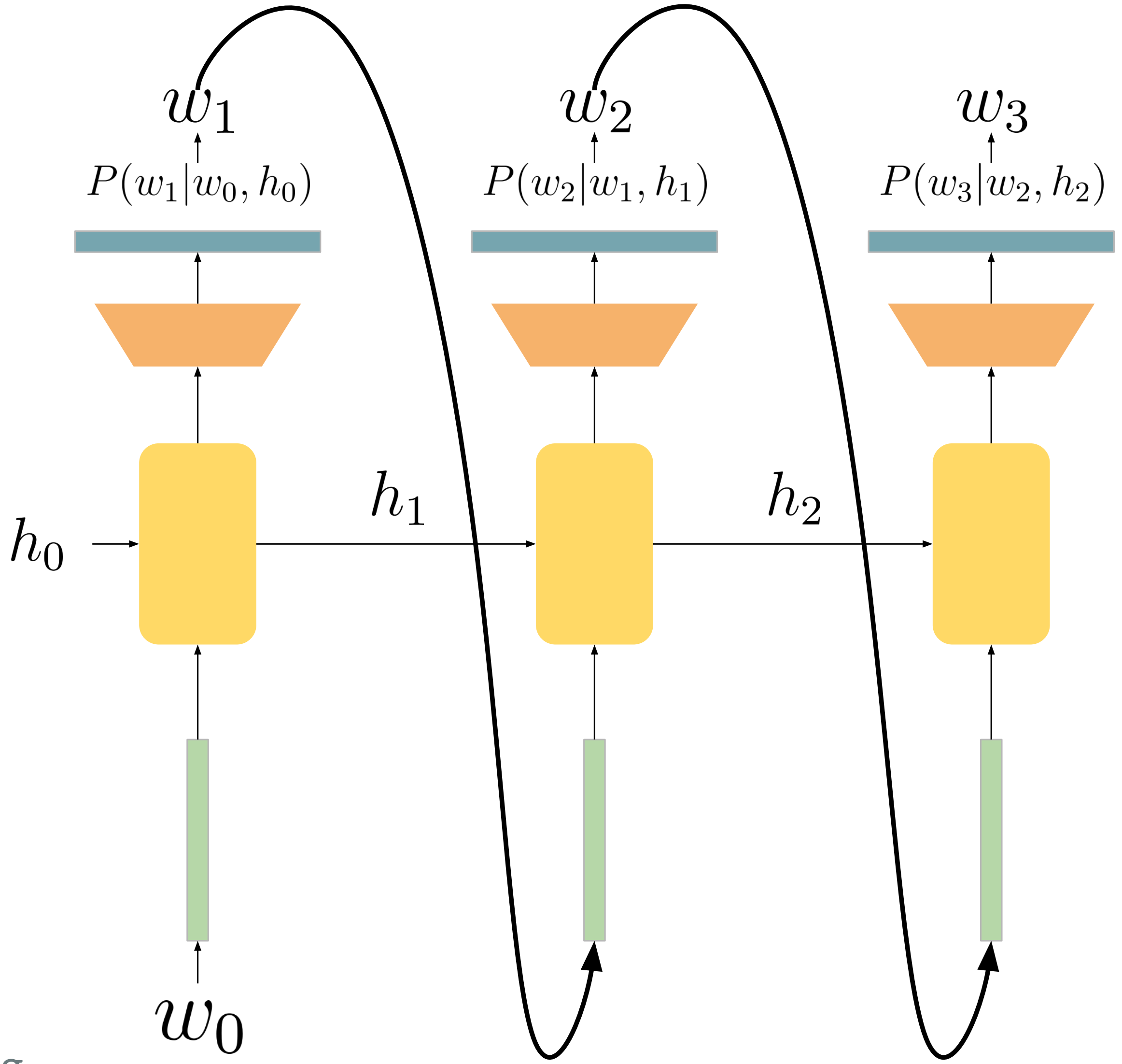
Reminder - Back-propagation



Reminder - Stochastic Gradient Descent

- **Goal** : Minimize a loss function $\mathcal{L}(X, \theta)$ for given data X with respect to model parameters θ
- **Method** :
  - Split X in smaller parts x^i (called mini-batches)
  - Compute $\mathcal{L}(x^i, \theta)$ (forward) and $\nabla_\theta \mathcal{L}(x^i, \theta)$ (back-prop)
  - Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(x^i, \theta)$ $\qquad$ $(\eta \ll 1$, learning rate)

LM with RNNs: Generation

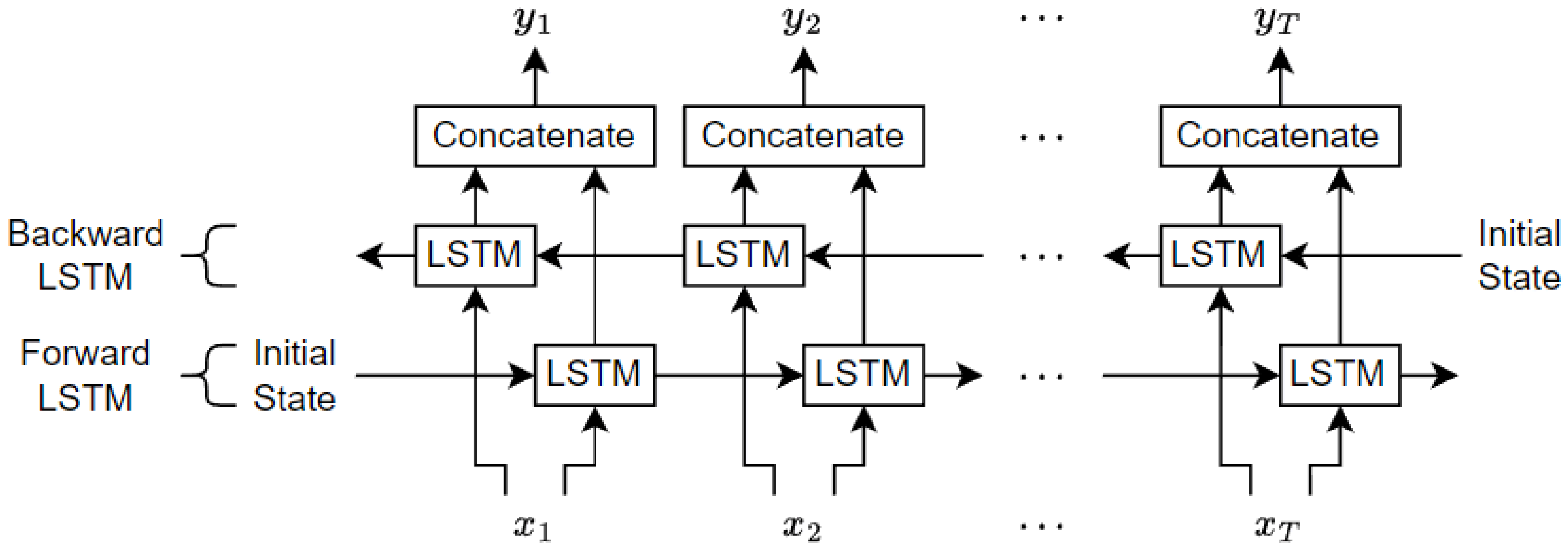


RNNs: pro/cons

- Strenghts
  - Still relatively fast to train
  - ... and for inference ($O(L)$)
  - **Can extrapolate** (works with continuous features)
- Limitations
  - **Context dilution** when information is far away

Extending RNNs: BiLSTMs

- LSTM: improves context capacity
- Read the sequence in both directions



Transformers

Information flow - RNN

How many steps between source of info and current position?

- *What is the previous word?* => $O(L)$
- *What is the subject of verb X?* => $O(L)$
- *What are the other occurences of current word?* => $O(L^2)$
- ...

Information flow - Transformers

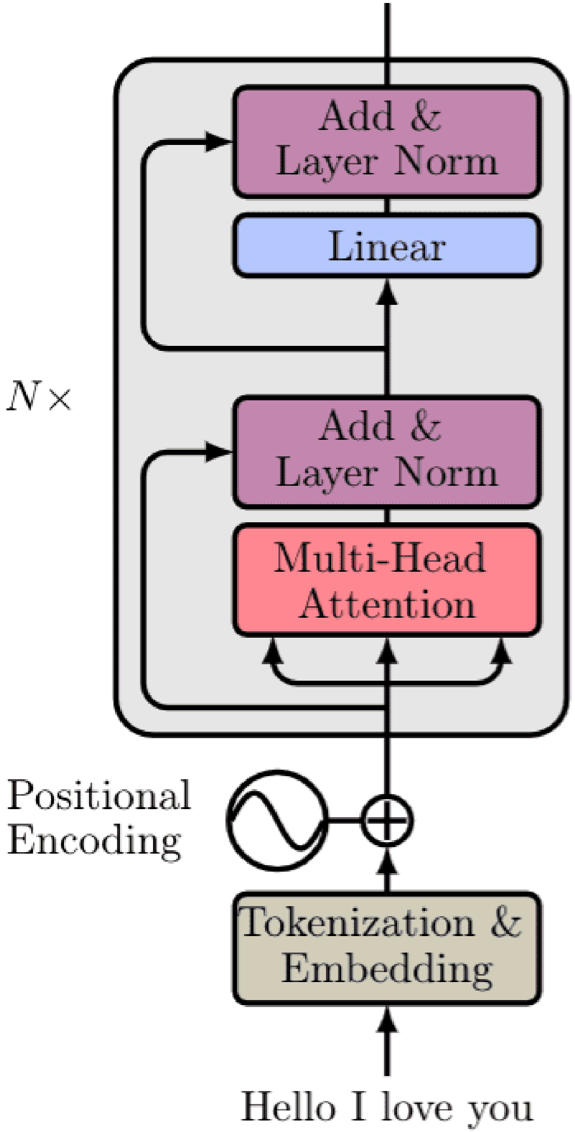
How many steps between source of info and current position?

- *What is the previous word?* => $O(1)$
- *What is the subject of verb X?* => $O(1)$
- *What are the other occurences of current word?* => $O(1)$
- ... => $O(1)$

Outside Transformers

- A Transformer network T_θ
- Input: Sequence of vectors $(e_1, \ldots, e_n) \in \mathbb{R}^D$
- Output: Sequence of vectors $(h_1, \ldots, h_n) \in \mathbb{R}^D$
- Each h_i may depend on the <u>whole</u> input sequence $(e_1, \ldots, e_n)$

Inside Transformers

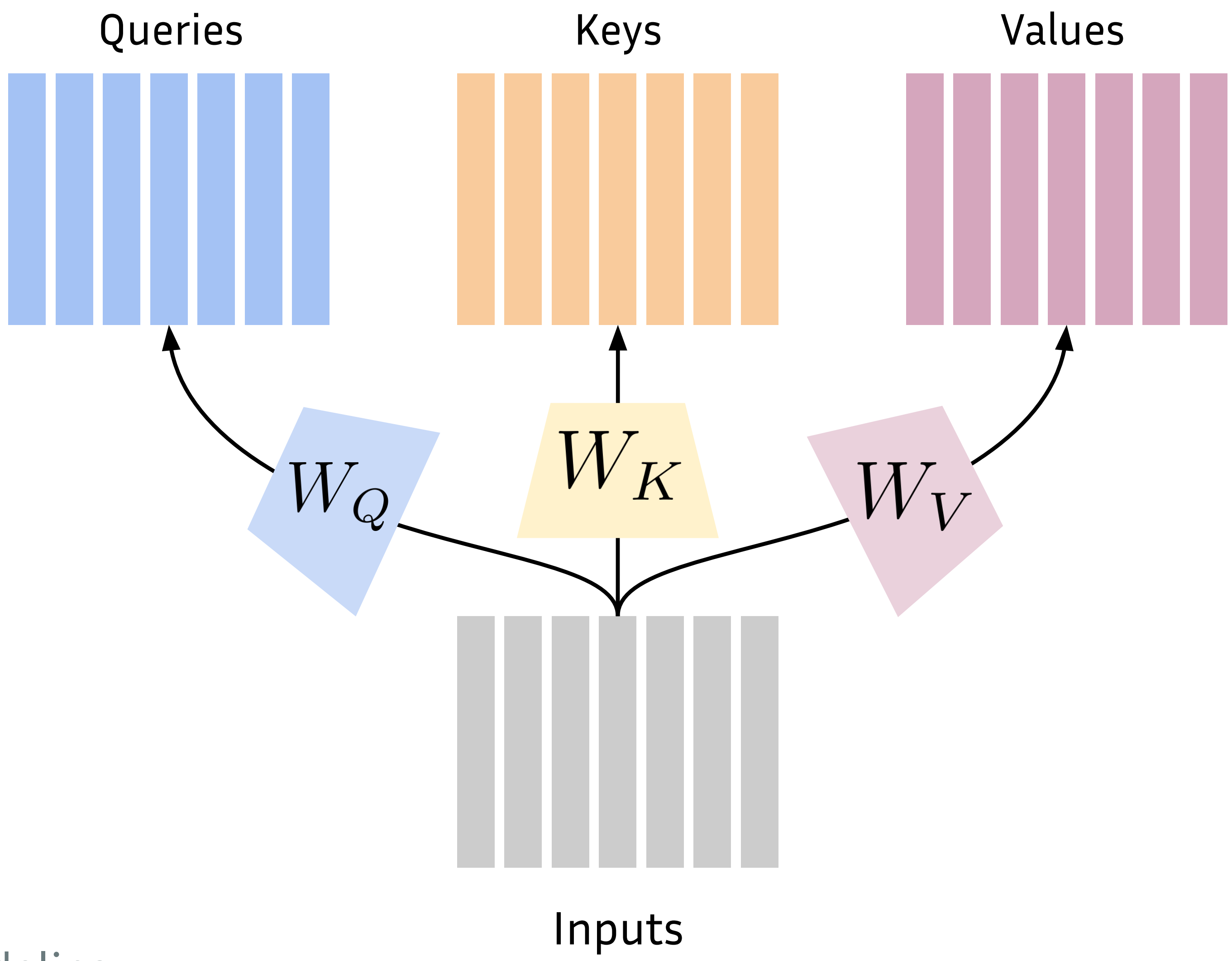


Inside Transformers : Embeddings

Before going in the network:

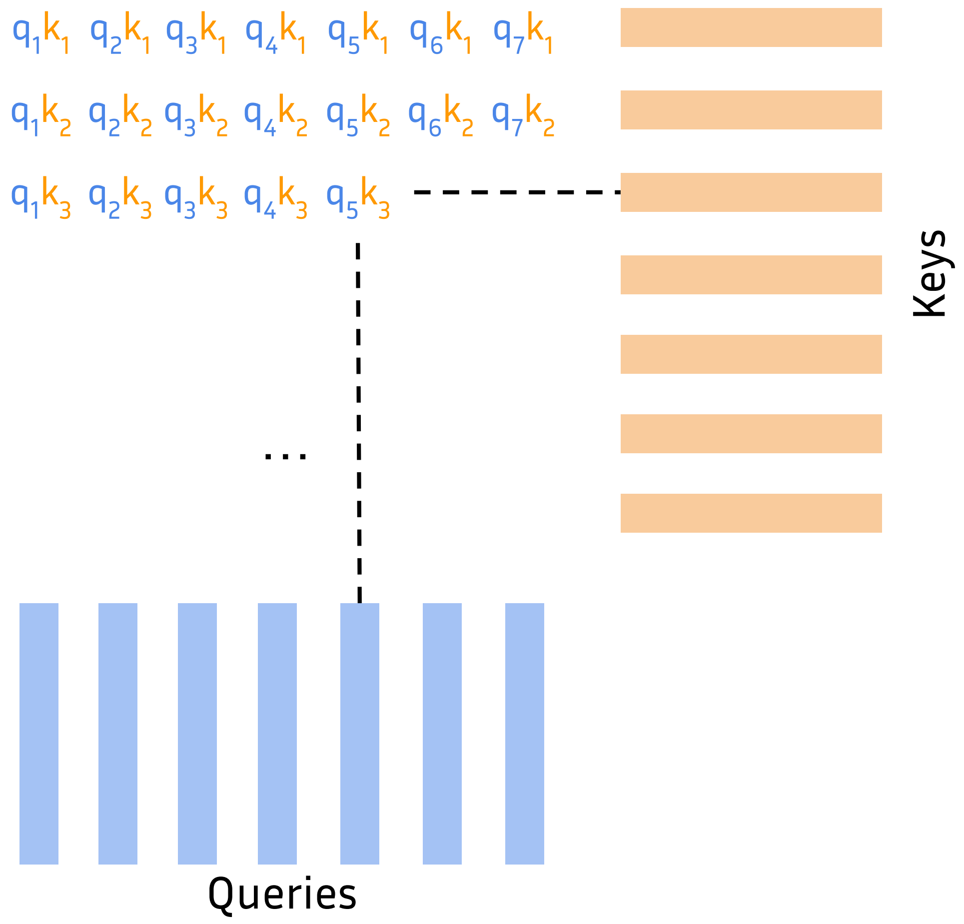
- Given an input token sequence $(w_1, \ldots, w_n)$
- We retrieve token embeddings $(e_w(w_1), \ldots, e_w(w_n)) \in \mathbb{R}^D$
- We retrieve position embeddings $(e_p(1), \ldots, e_p(n)) \in \mathbb{R}^D$
- We compute input embeddings: $e_i = e_w(w_i) + e_p(i)$

Inside Transformers : Self-attention



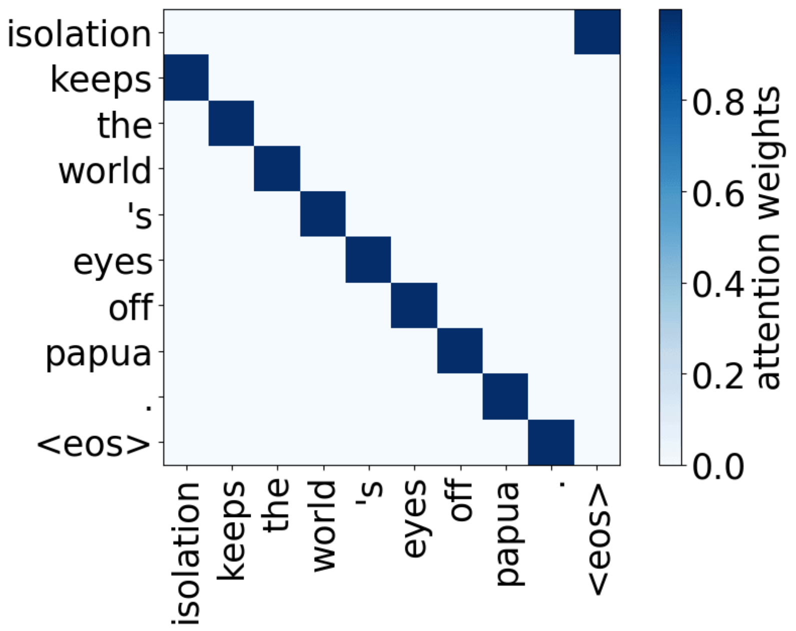
Inside Transformers : Q and K

=> Model interactions between tokens:



Inside Transformers : Q and K

- Each row of QK^T is then normalized using softmax
- Interpretable patterns:



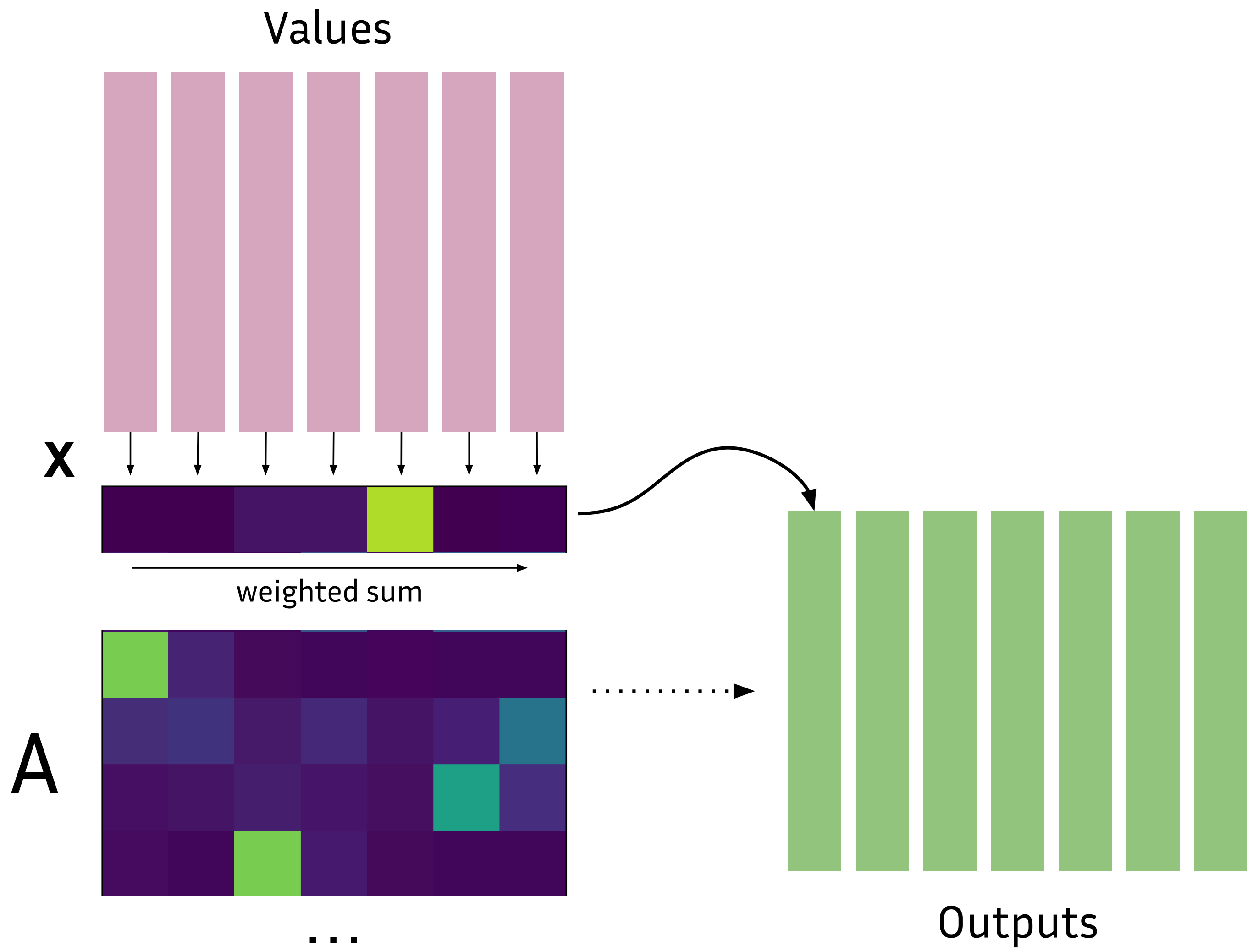
Inside Transformers : Q and K

- Formally:

$$A_{i,j} = \frac{1}{\sqrt{d_h}} \cdot \frac{e^{(QK^T)_{i,j}}}{\sum_k e^{(QK^T)_{i,k}}}$$

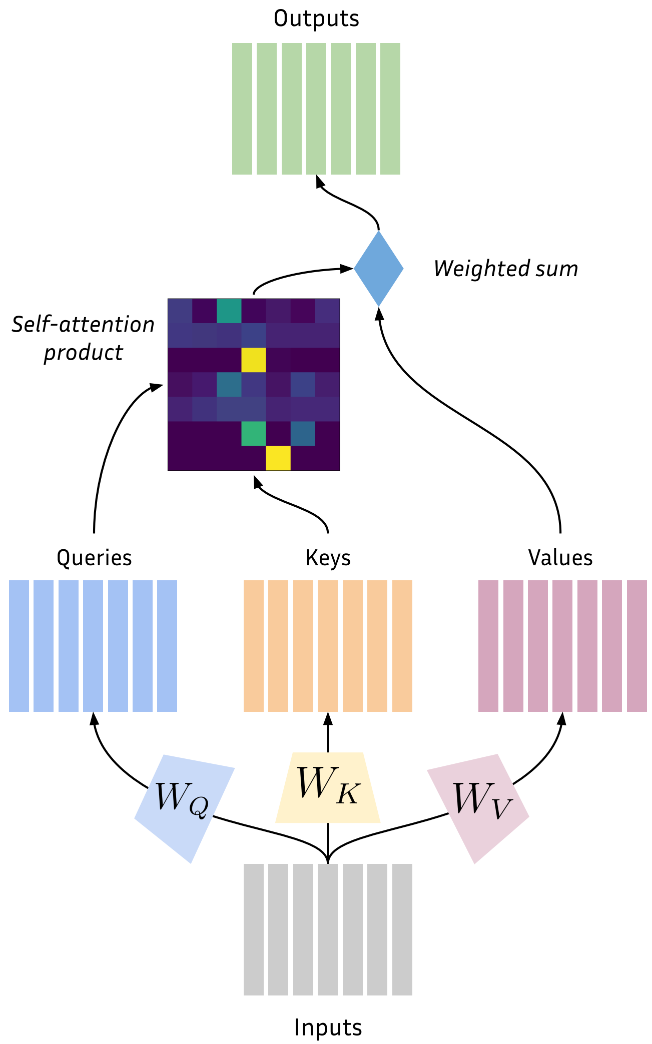
where d_h is the hidden dimension of the model

Inside Transformers : A and V

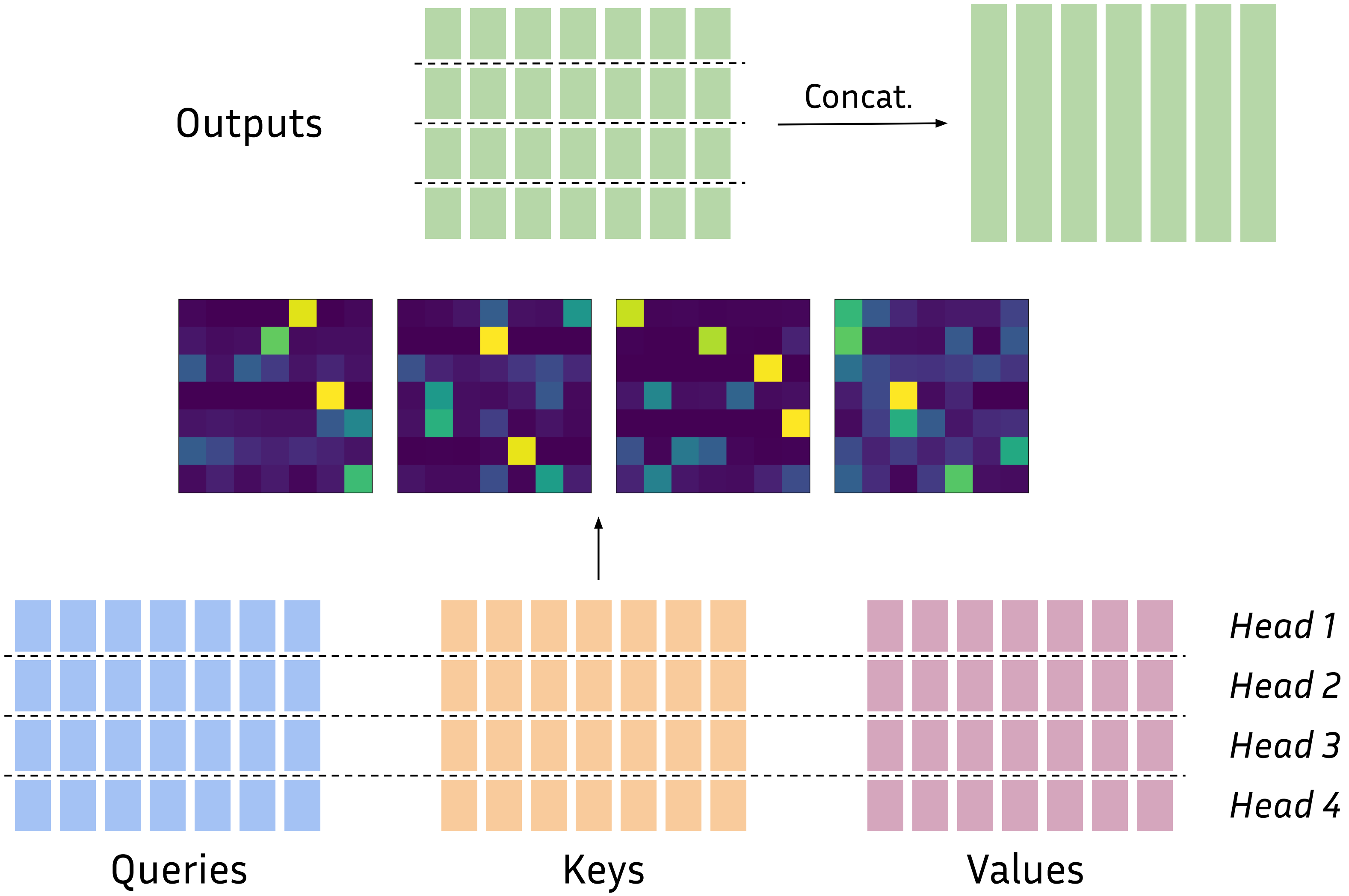


Inside Transformers : Self-attention summary

- Inputs are mapped to Queries, Keys and Values
- Queries and Keys are used to measure interaction (A)
- Interaction weights are used to "select" relevant Values combinations
- **Complexity: O(L^2)**

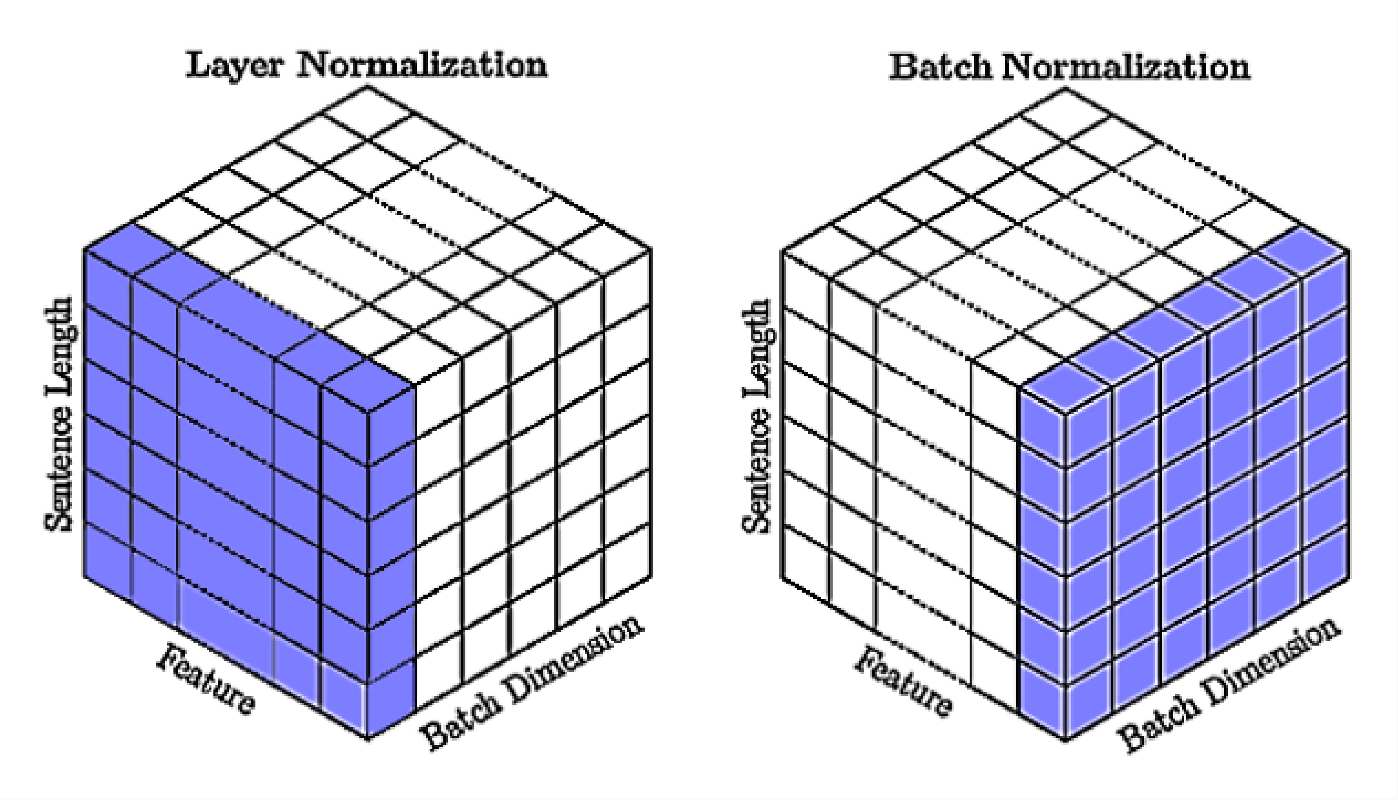


Inside Transformers : Multi-head attention

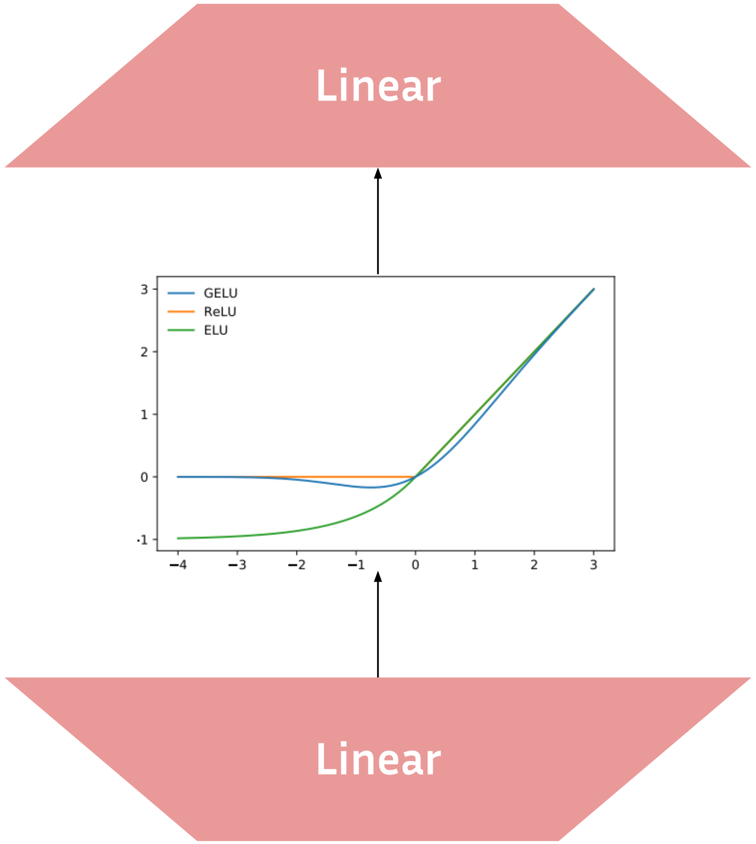


Inside Transformers : LayerNorm

- Avoids gradient explosion



Inside Transformers : Output layer



Modern flavors : Relative Positional Embeddings

- Encode position at attention-level:

$$(\Omega QK^T)_{i,j} = \langle \omega_i(Q_i), \omega_j(K_j) \rangle + \beta_{i,j}$$

- Rotary Positional Embeddings (RoPE, Su et al. 2023)
  - ω_i is a rotation of angle $i\theta$; no β
- Linear Biases (ALiBi, Press et al. 2022)
  - $\beta_{i,j} = m \cdot (i - j)$ with $m \in \mathbb{R}$

Modern flavors : RMSNorm

- Replaces LayerNorm
- Re-scaling is all you need

$$RMSNorm_g(a_i) = \frac{a_i}{\sqrt{\frac{1}{N}\sum_{j=1}^{N} a_j^2}} g_i$$

Modern flavors : Grouped-Query Attention

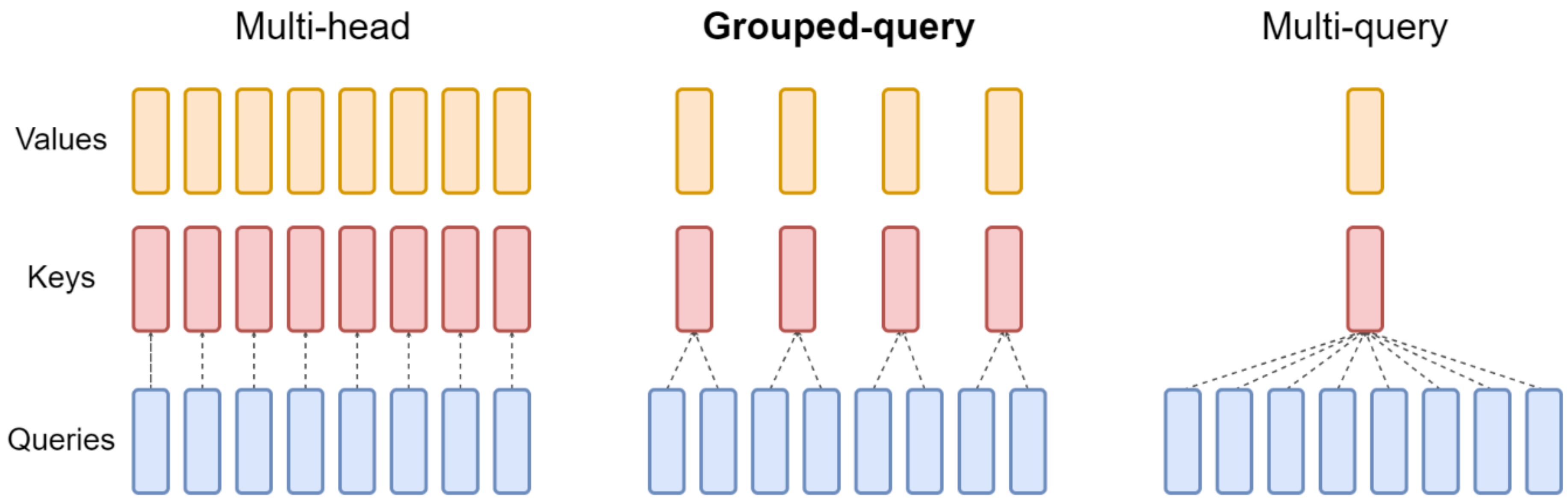
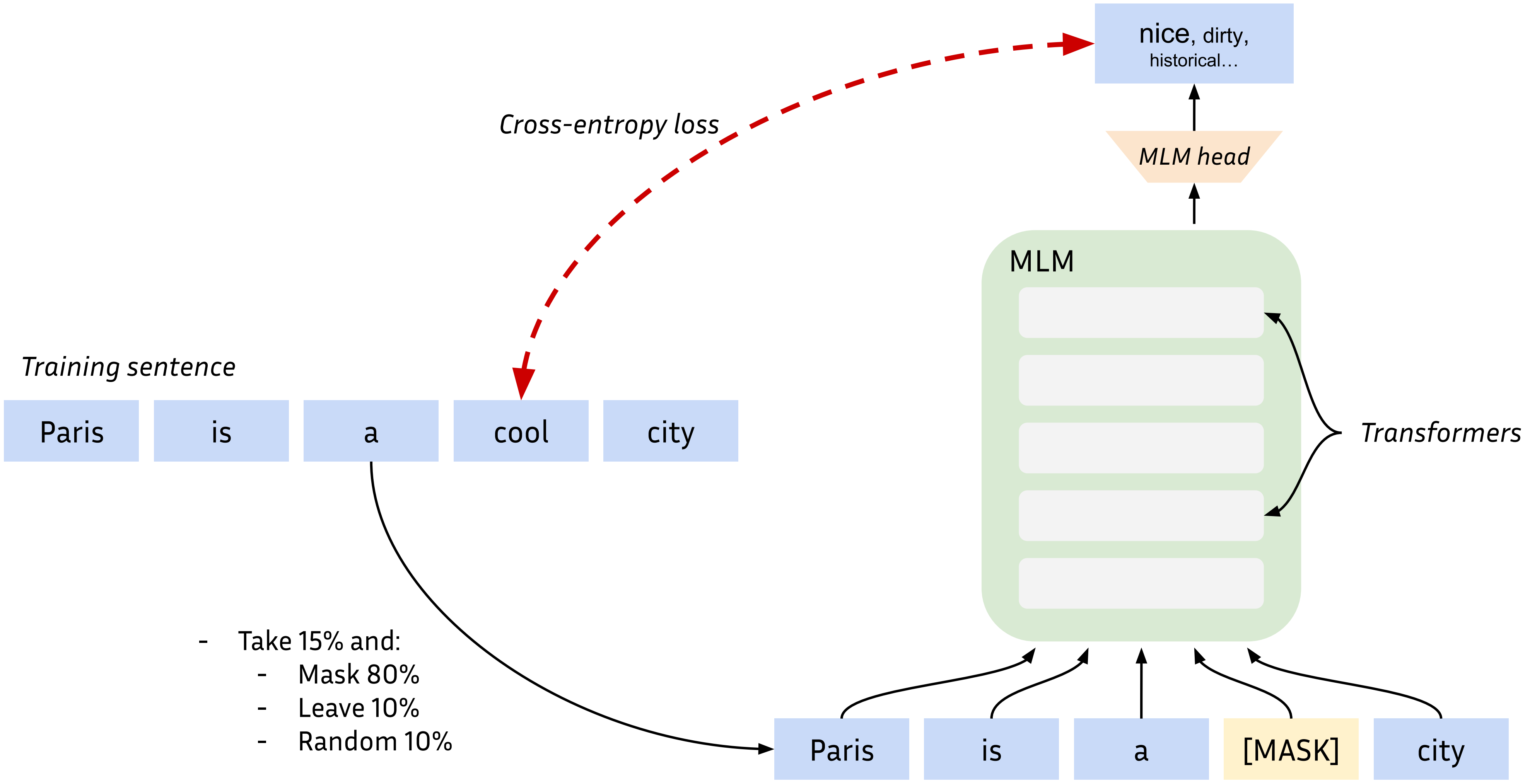


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Encoder Models

Masked Language Models



- Take 15% and:
    - Mask 80%
    - Leave 10%
    - Random 10%

BERT (Devlin et al., 2018)

- Pre-trained on 128B tokens from Wikipedia + BooksCorpus
- Additional Next Sentence Prediction (NSP) loss
- Two versions:
  - BERT-base (110M parameters)
  - BERT-large (350M parameters)
- **Cost**: ~1000 GPU hours

RoBERTa (Liu et al., 2019)

- Pre-trained on ~~128B~~ **2T** tokens from web data (BERT x10)
- **No more** Next Sentence Prediction (NSP) loss
- Two versions:
  - RoBERTa-base (110M parameters)
  - RoBERTa-large (350M parameters)
- Better results in downstream tasks
- **Cost**: ~25000 GPU hours

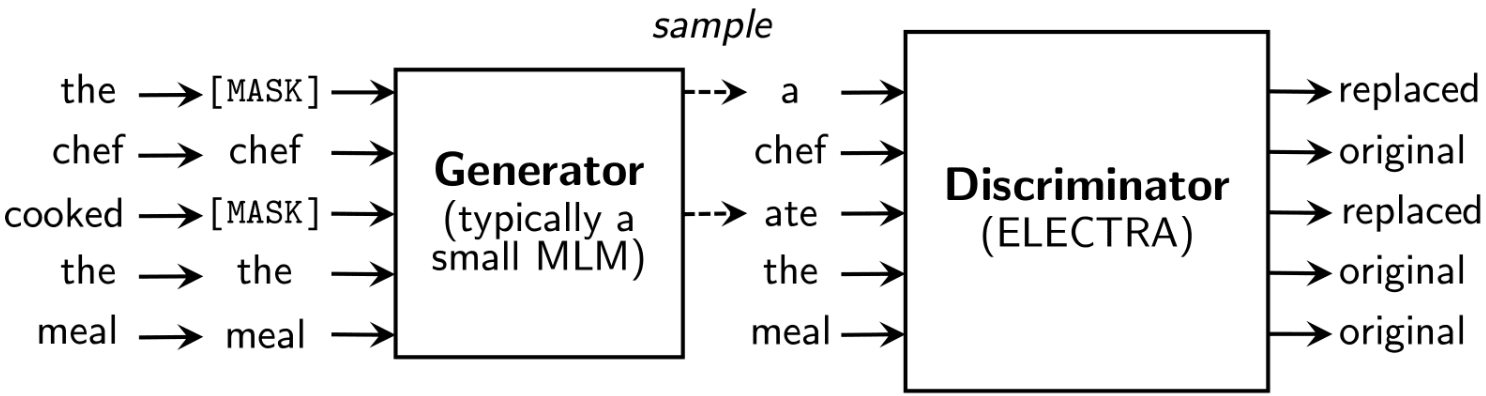
Multilingual BERT (mBERT)

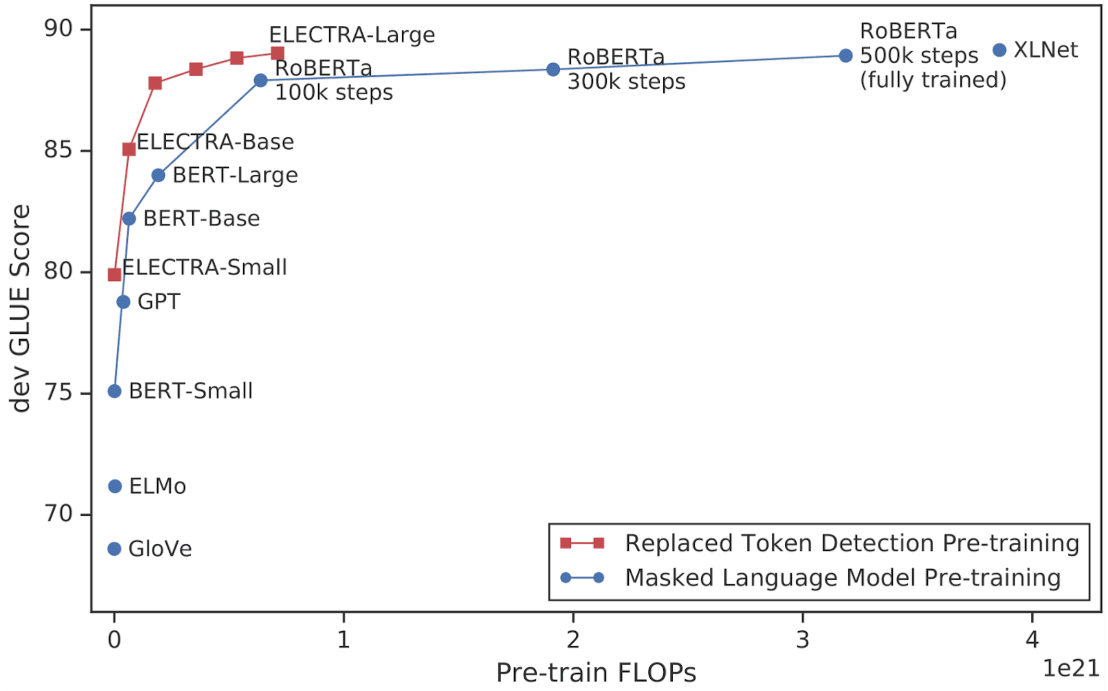
- Pre-trained on 128B tokens from multilingual Wikipedia
- 104 languages
- One version:
  - mBERT-base (179M parameters)
- **Cost**: *unknown*

XLM-RoBERTa (Conneau et al., 2019)

- Pre-trained on **63T** tokens from CommonCrawl
- 100 languages
- Two versions:
  - XLM-RoBERTa-base (279M parameters)
  - XLM-RoBERTa-large (561M parameters)
- **Cost**: ~75000 GPU hours

ELECTRA (Clark et al., 2020)

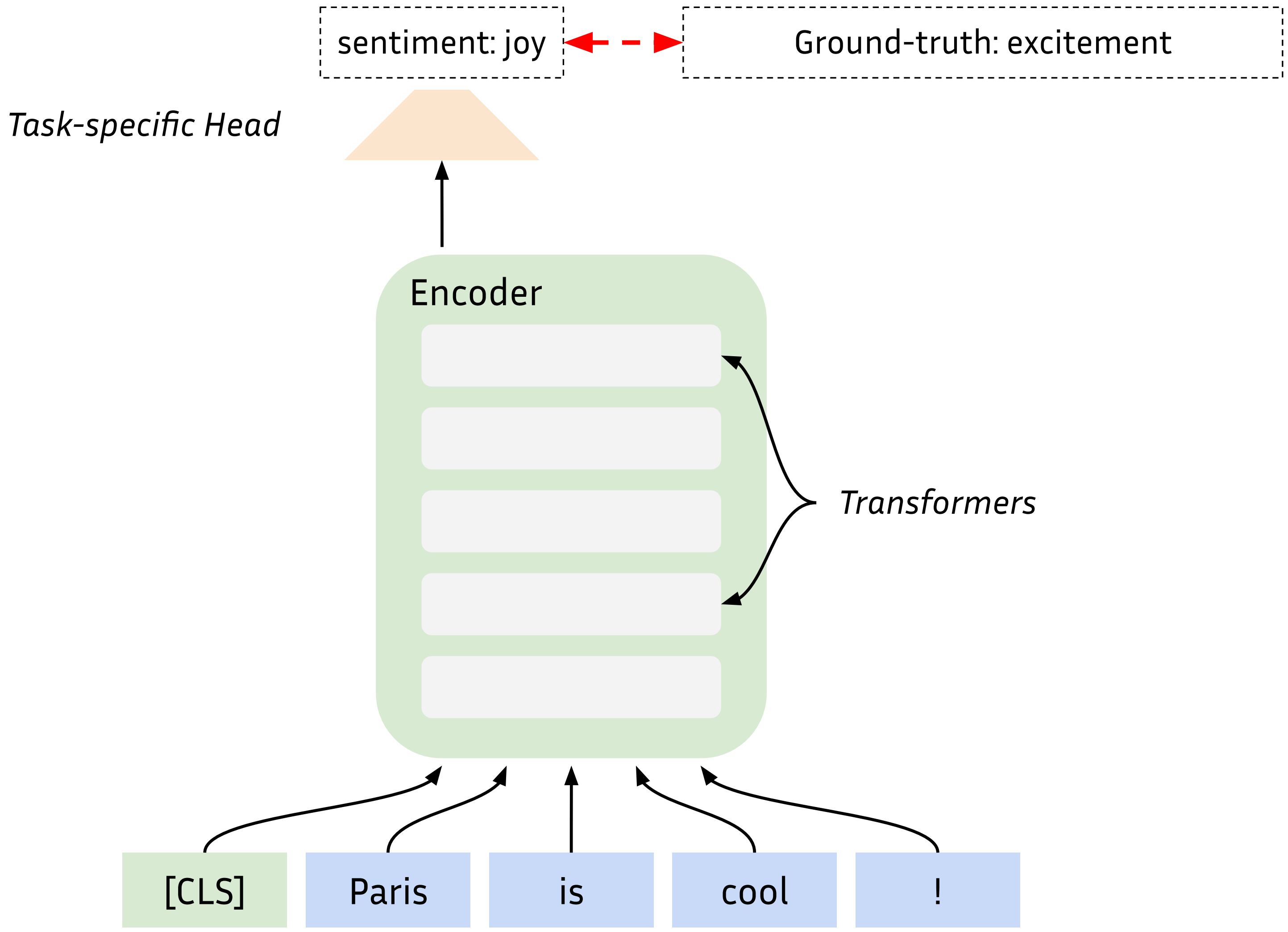




ELECTRA (Clark et al., 2020)

- Pre-trained on **63T** tokens from CommonCrawl
- 100 languages
- Three versions:
  - ELECTRA-small (14M parameters)
  - ELECTRA-base (110M parameters)
  - ELECTRA-large (350M parameters)
- Really better than BERT/RoBERTa
- **Cost**: =BERT

Encoders: Fine-tuning



Encoders: Classical applications

- Natural Language Inference (NLI)
  - *I like cake! / Cake is bad* => ~~same~~|~~neutral~~|**opposite**
- Text classification (+ clustering)
  - *I'm so glad to be here!* => joy
- Named Entity Recognition (NER)
  - *I voted for Obama!* => (Obama, pos:3, class:PER)
- and many others...

Decoders

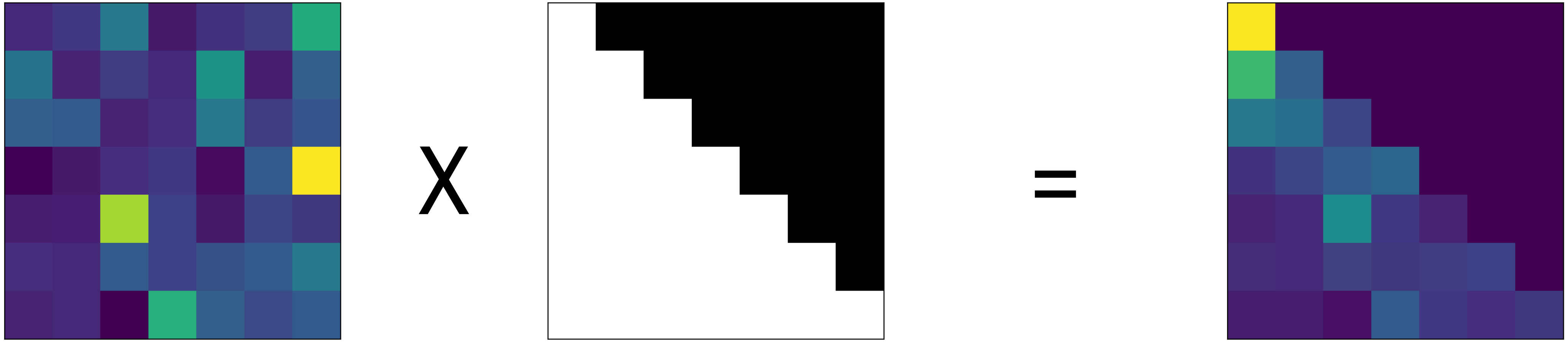
Decoders - Motivation

- Models that are designed to **generate text**
- Next-word predictors:

$$P(w_i \,|(w_j)_{j\neq i}) = P_\theta(w_i|w_1 \dots w_{i-1})$$

- **Problem**: How do we impede self-attention to consider future tokens?

Decoders - Attention mask

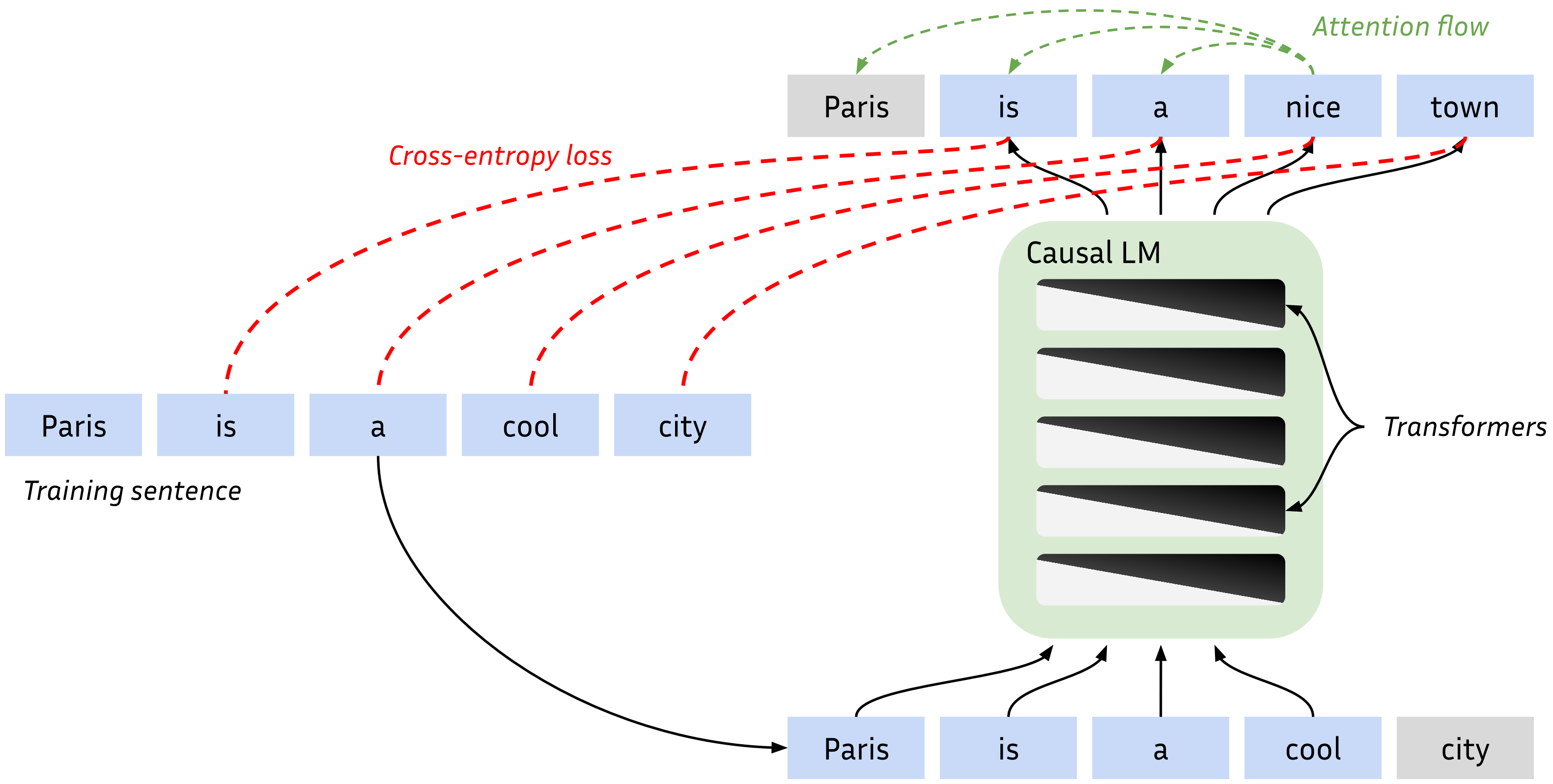


Attention *Attention mask* ***Causal** attention*

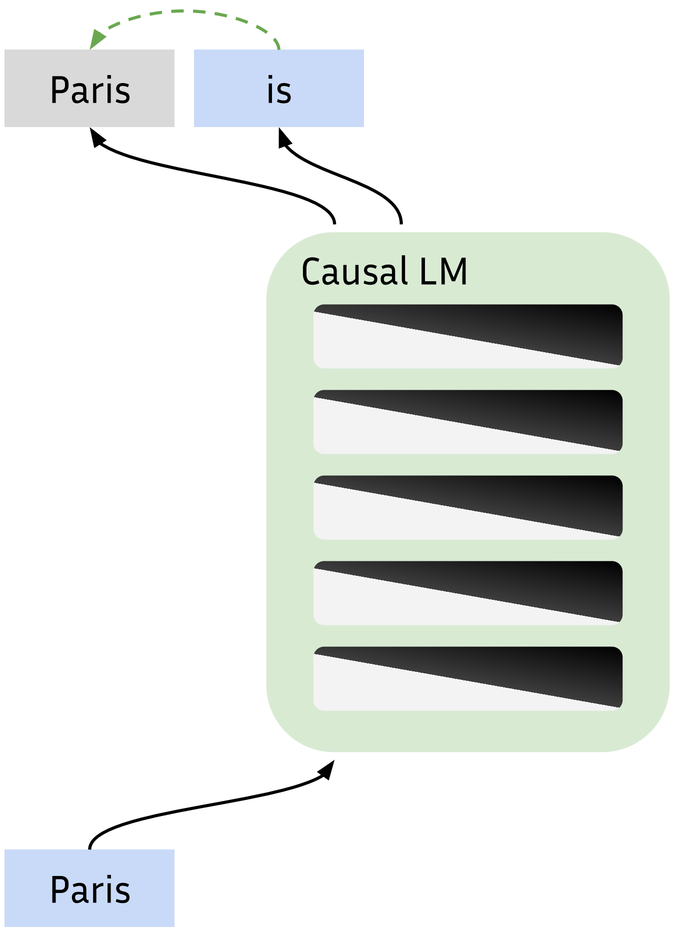
- Each attention input can only attend to previous positions

Decoders - Causal LM pre-training

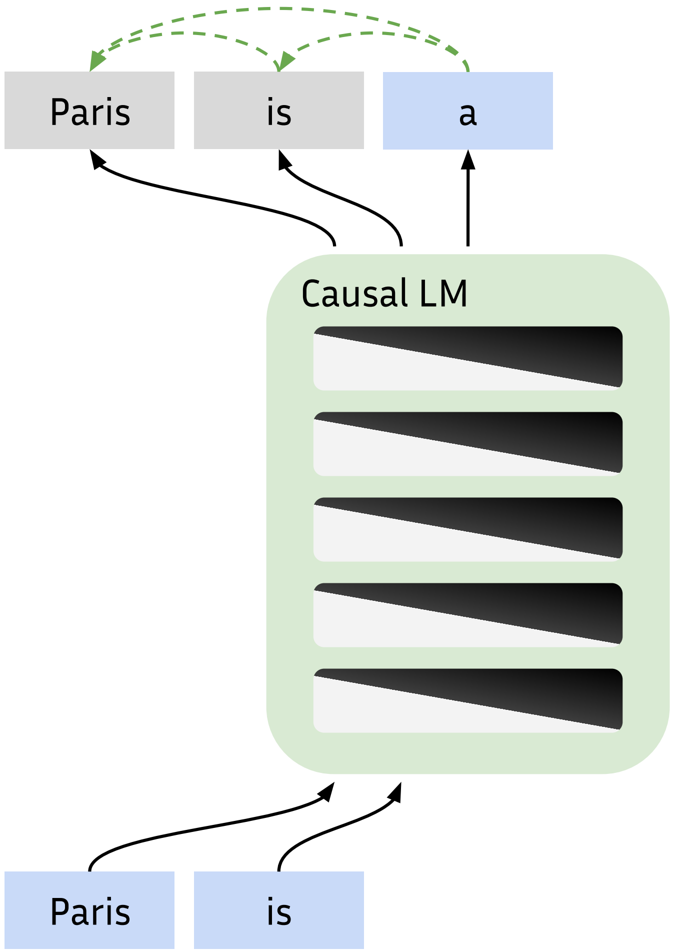
- Teacher-forcing



Decoders - Causal LM inference (greedy)



Decoders - Causal LM inference (greedy)



Decoders - Refining inference

- What we have : a good model for $P_\theta(w_i|w_1 \dots w_{i-1})$
- What we want at inference:

$$W^* = \backslash\mathbf{argmax}_{n,w_i \dots w_n} P_\theta(w_i \dots w_n|w_1 \dots w_{i-1})$$

- For a given completion length n, there are $|V|^n$ possibilities
  - e.g.: 19 new tokens with a vocab of 30000 tokens > #atoms in Ω
- We need approximations

Decoders - Greedy inference

- Keep best word at each step and start again:

$$W^* = \backslash\mathrm{argmax}_{n,w_{i+1}\dots w_n} P_\theta(w_{i+1}\dots w_n | w_1 \dots w_{i-1} w_i^*)$$

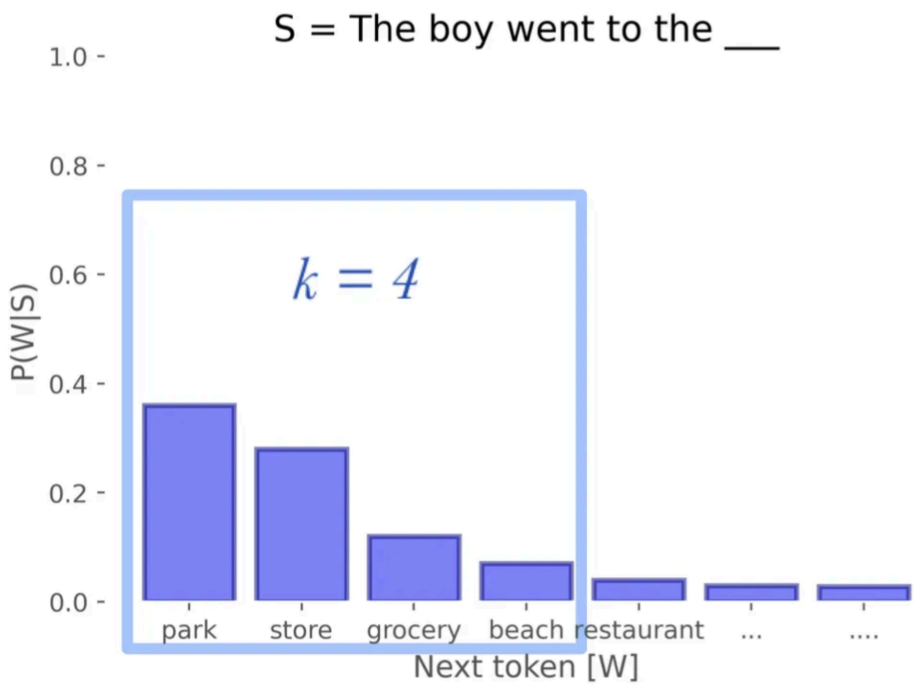
where $w_i^* = \backslash\mathrm{argmax}_{w_i} P_\theta(w_i | w_1 \dots w_{i-1})$

Decoders - Beam search

- Keep best k chains of tokens at each step:
  - Take k best w_i and compute $P_\theta(w_{i+1}|\ldots w_i)$ for each
  - Take k best w_{i+1} in each sub-case (now we have $k \times k$ (w_i, w_{i+1}) pairs to consider)
  - Consider only the k more likely (w_i, w_{i+1}) pairs
  - Compute $P_\theta(w_{i+2}|\ldots w_i w_{i+1})$ for the k candidates
  - and so on...

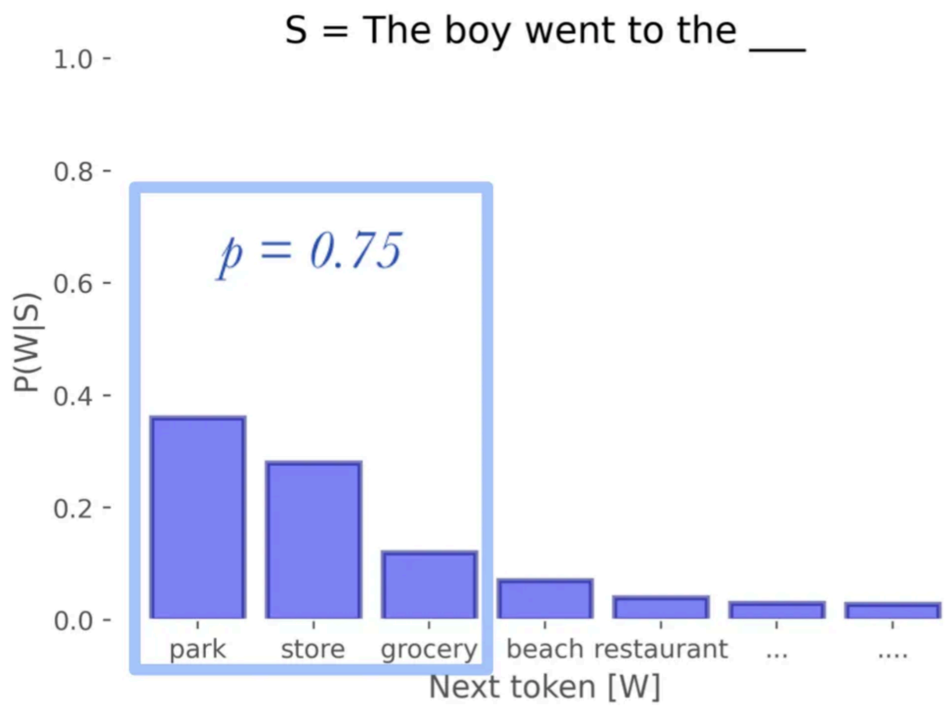
Decoders - Top-k sampling

- Randomly sample among top-k tokens based on P_θ



Decoders - Top-p (=Nucleus) sampling

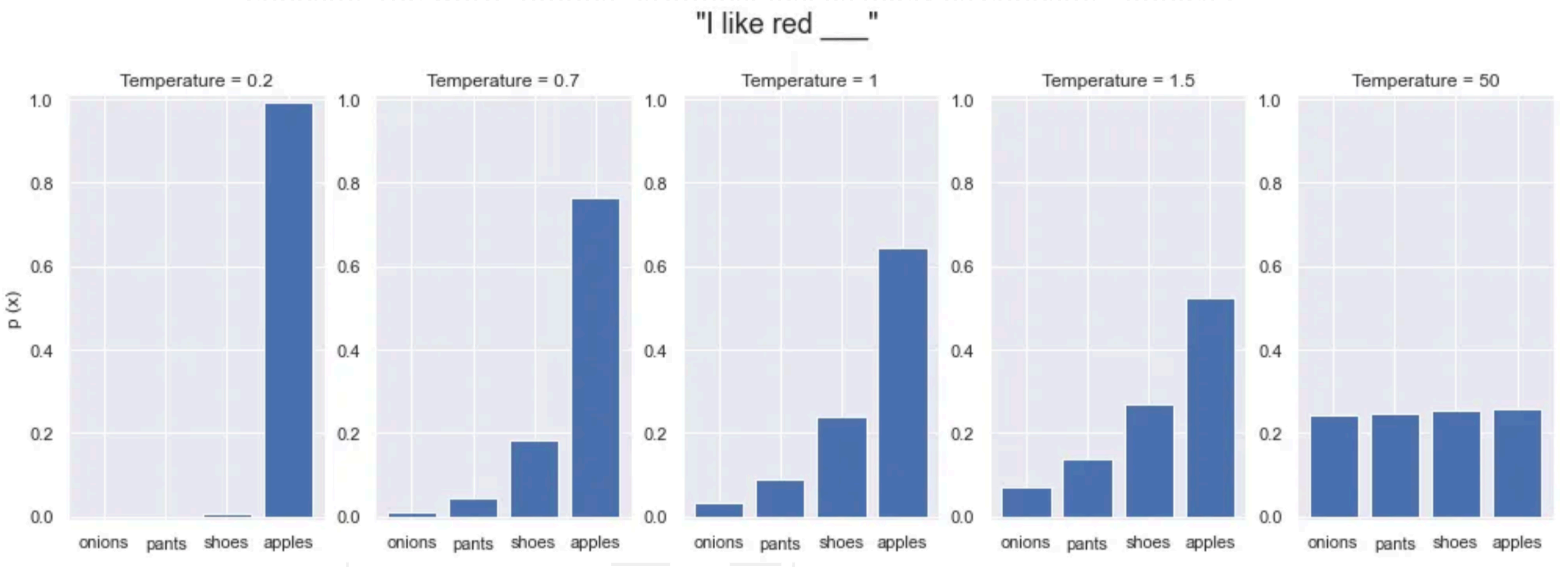
- Randomly sample based on P_θ up to $p\%$



Decoders - Generation Temperature

- Alter the softmax function:

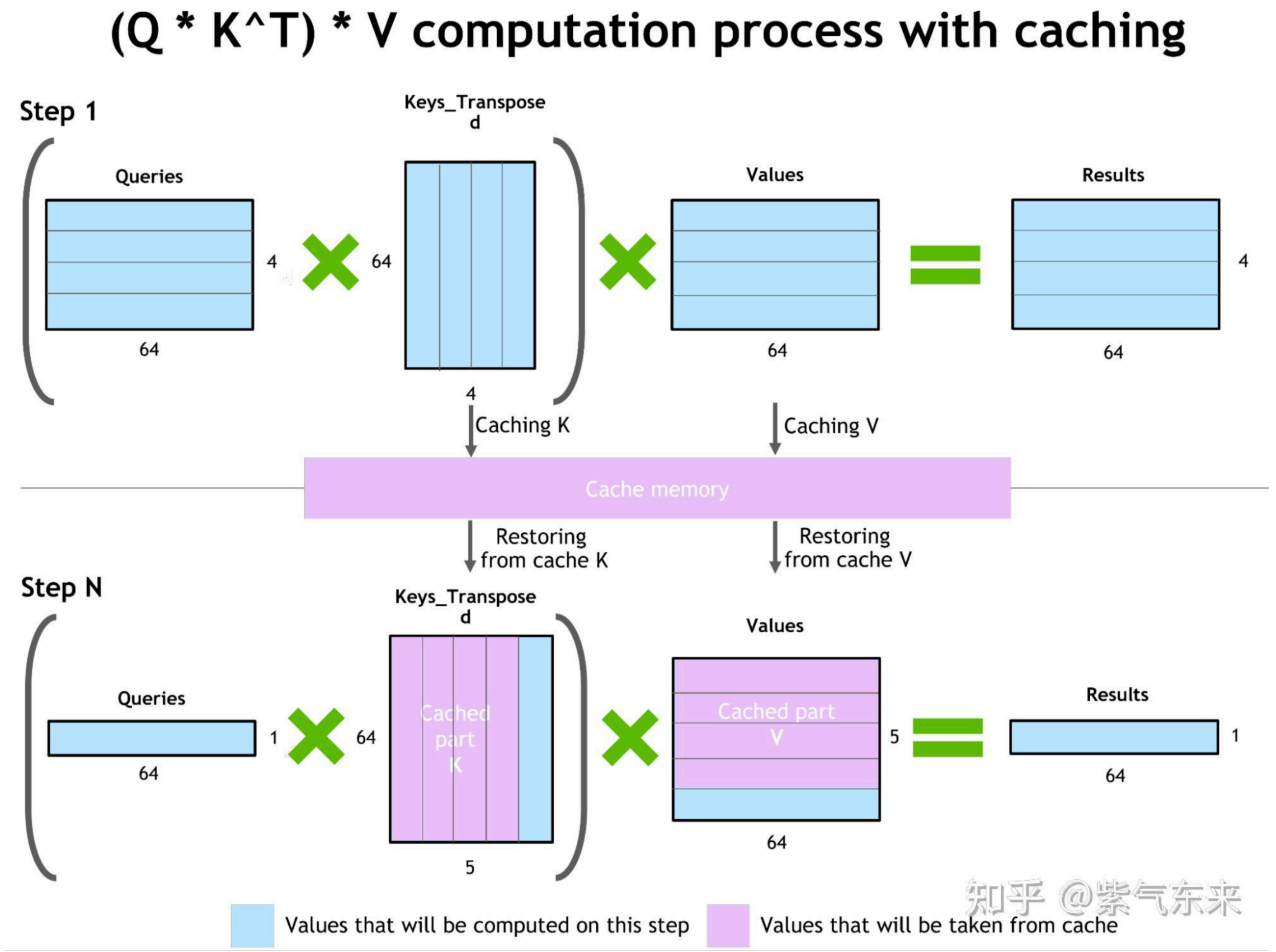
$$softmax_\tau(x) = \frac{e^{\frac{x_i}{\tau}}}{\sum_j e^{\frac{x_j}{\tau}}}$$



Decoders - Inference speed

- For greedy decoding without prefix:
  - n passes with sequences of length $1 \leq t \leq n$
  - Each pass is $O(n^2)$
  - Complexity: $O(n^3)$
- Other decoding are more costly
- Ways to go faster?

Decoders - Query-Key caching

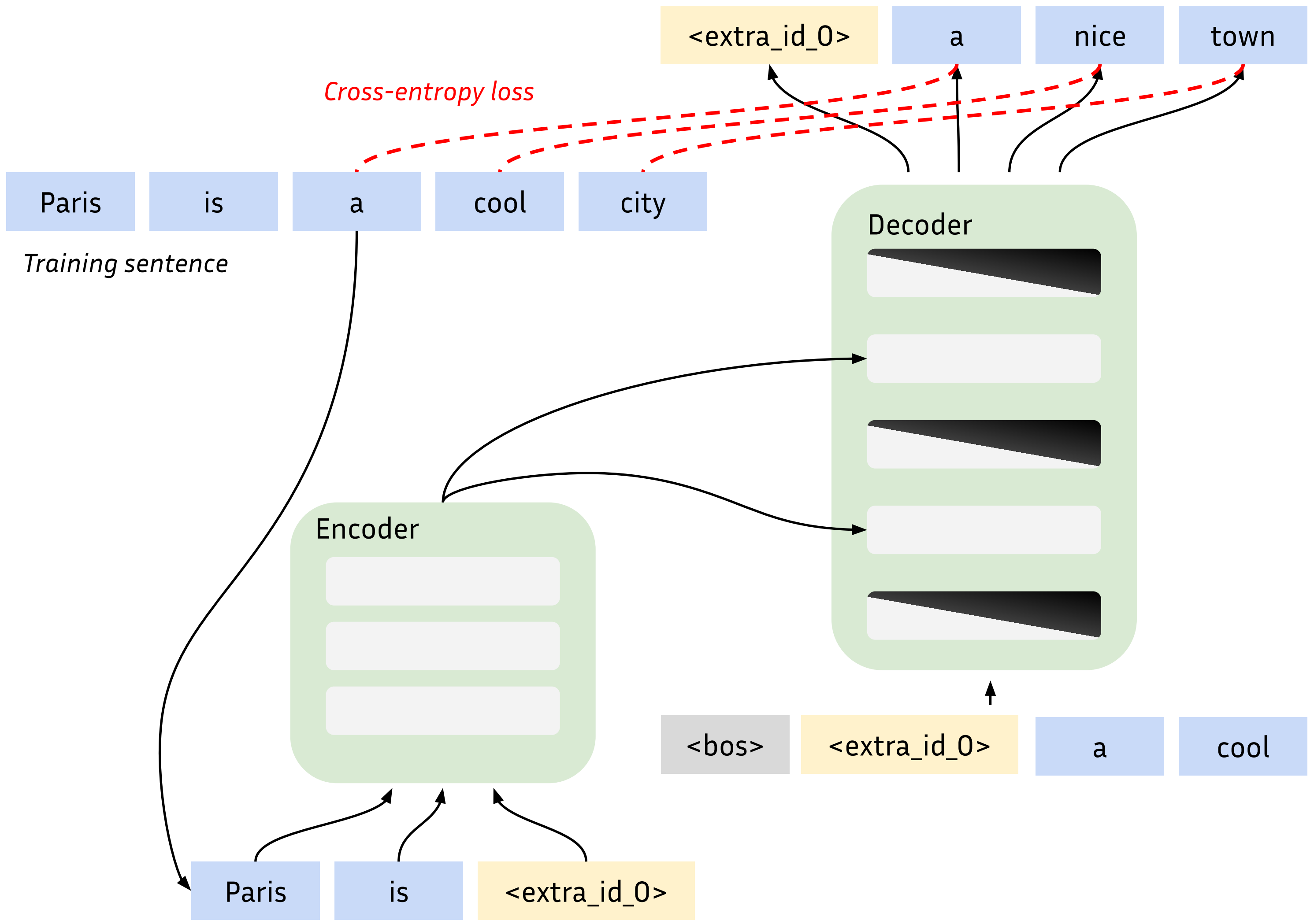


Decoders - Speculative decoding

- Generate γ tokens using P_ϕ where $|\phi| \ll |\theta|$ (smaller model)
- Forward $w_i \dots w_{i+\gamma}$ in teacher-forcing mode and predict $w_{i+\gamma+1}$ with the bigger model
- Compare P_θ and P_ϕ and only keep tokens where they don't differ too much

Encoder-Decoder models

T5 pre-training



All models can do everything

- Encoders are mostly used to get contextual embeddings
  - They can also generate : T_{enc}("I love [MASK]")
- Decoders are mostly used for language generation
  - They can also give contextual embeddings : T_{dec}("I love music!")
  - Or solve any task using prompts:
    - "What is the emotion in this tweet? Tweet: '...' Answer:"
- Encoders-decoders are used for language in-filling

Evaluating models

- A useful evaluation metric: ***Perplexity***
- Defined as:

$$ppl(T_\theta; w_1 \dots w_n) = \exp\left(-\frac{1}{n}\sum_{t=1}^{n}\log P_\theta(w_t | w_{<t})\right)$$

- Other metrics: accuracy, MAUVE, ...

Zero-shot evaluation

- Never-seen problems/data
- Example: *"What is the capital of Italy? Answer:"*
    - Open-ended: Let the model continue the sentence and check exact match
    - Ranking: Get next-word likelihood for *"Rome"*, *"Paris"*, *"London"*, and check if *"Rome"* is best
    - Perplexity: Compute perplexity of *"Rome"* and compare with other models

Few-shot evaluation / In-context learning

- Never-seen problems/data
- Example: *"Paris is the capital of France. London is the capital of the UK. Rome is the capital of"*
- Chain-of-Thought (CoT) examples:
    - Normal: *"(2+3)x5=25. What's (3+4)x2?"*
    - CoT: *"To solve (2+3)x5, we first compute (2+3) = 5 and then multiply (2+3)x5=5x5=25. What's (3+4)x2?"*

Open-sourced evaluation

- Generative models are evaluated on benchmarks
- Example (LLM Leaderboard from HuggingFace):

T	Model	Average	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K
🟢	Qwen/Qwen-72B	73.6	65.19	85.94	77.37	60.19	82.48	70.43
🟢	chargoddard/Yi-34B-Llama	70.95	64.59	85.63	76.31	55.6	82.79	60.8
🟢	01-ai/Yi-34B-200K	70.81	65.36	85.58	76.06	53.64	82.56	61.64
🟢	01-ai/Yi-34B	69.42	64.59	85.69	76.35	56.23	83.03	50.64
🟢	deepseek-ai/deepseek-llm-67b-base	69.38	65.44	87.1	71.78	51.08	84.14	56.71
🟢	mistralai/Mixtral-8x7B-v0.1	68.42	66.04	86.49	71.82	46.78	81.93	57.47
🟢	meta-llama/Llama-2-70b-hf	67.87	67.32	87.33	69.83	44.92	83.74	54.06
🟢	tiiuae/falcon-180B	67.85	69.45	88.86	70.5	45.47	86.9	45.94

Lab session