COMP0087

Statistical Natural Language Processing

Lecture - Transformer Language Models, Pretraining, PEFT

Slides present original content as well as content based on what was previously developed by:

- Pascal Poupart
- Anna Goldie
- Ashish Vaswani
- Stanford CS224N

What reference text book to read?

The field is moving rapidly. You have lots of resources on the internet to explore pretty much any ideas and topics in AI.

But if you were to learn about classic and modern NLP systematically, Jurafsky & Martin is a fairly updated book:

https://web.stanford.edu/~jurafsky/slp3/

Blogs, Podcasts, Digests, Tools

Here are some of my suggestions for public commuters:

- Andrew Ng's The Batch weekly newsletters
- This Week in ML and AI [Spotify]
- Machine Learning Street Talk [Spotify, Youtube]
- HuggingFace Daily Paper Digest [HF]
- Or create your own podcast of the papers via NotebookLM: https://notebooklm.google/

Ollama is an open-source platform that enables users to run open source large language models (LLMs) directly on their local machines.

- Ollama official webpage: https://ollama.com
- GitHub repository: https://github.com/ollama/ollama

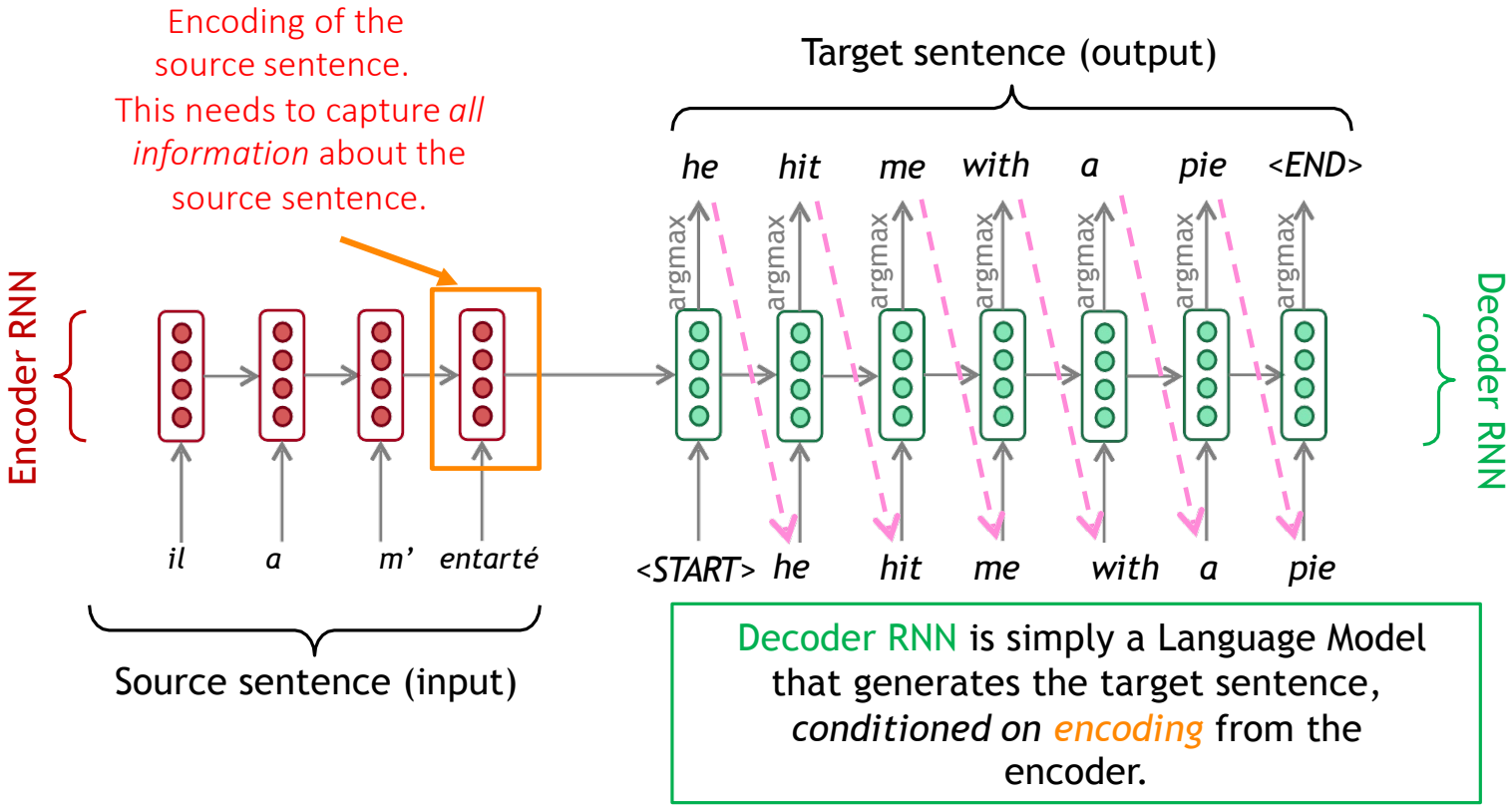
You can simply download your favourite model (e.g., Deepseek R1, gpt-oss, Qwen) and either directly use it offline as your chat assistant, or call it from python (i.e., see this).

Open WebUI is a user-friendly self-hosted AI platform designed to operate entirely offline. It supports various LLM runners like Ollama, and RAG.

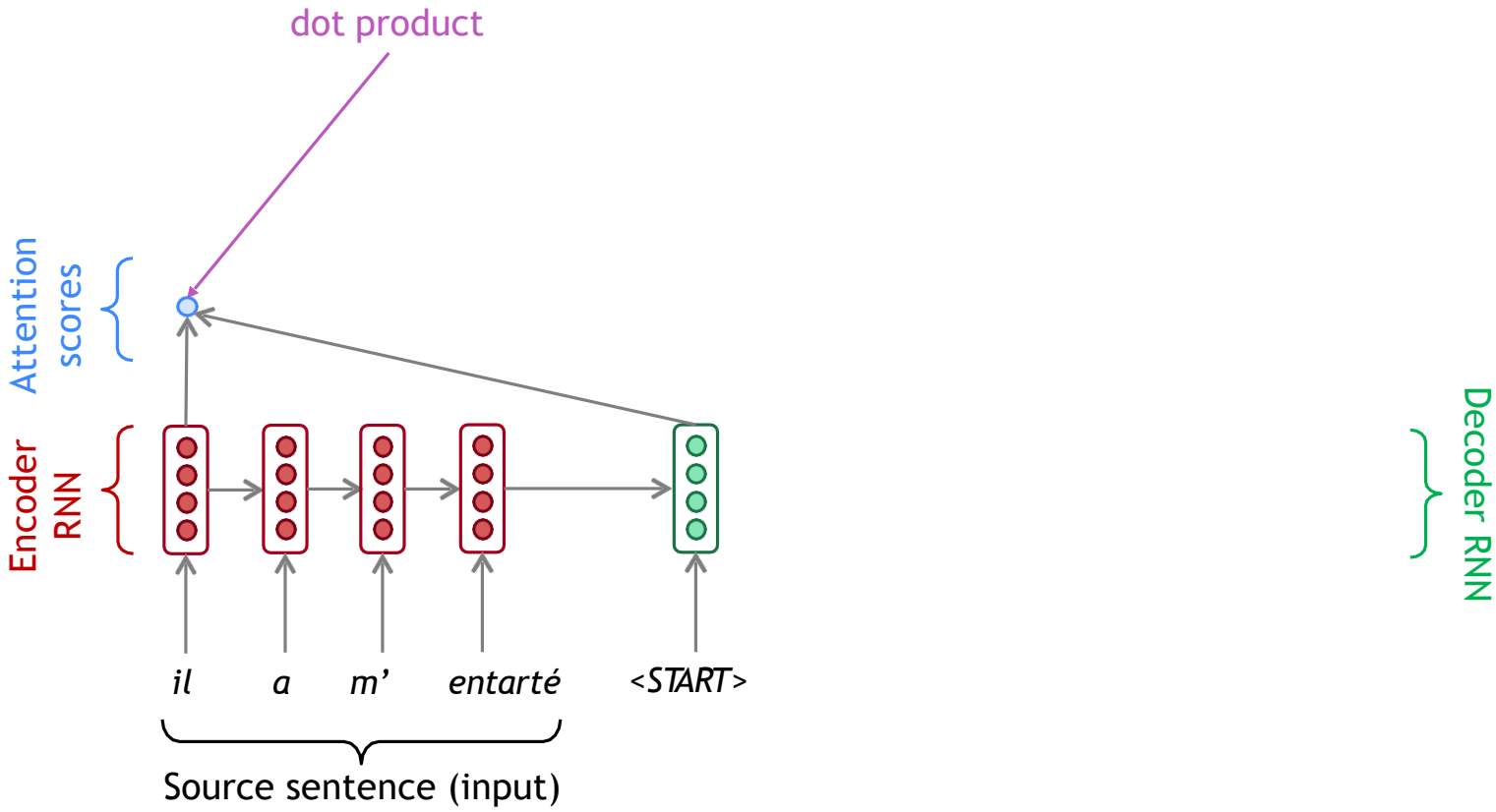
Overview

1. **Recap: Attention mechanism**
2. Attention as a lookup, and self-attention
3. Transformers
4. Transformer Language Models
5. Pretraining and Finetuning

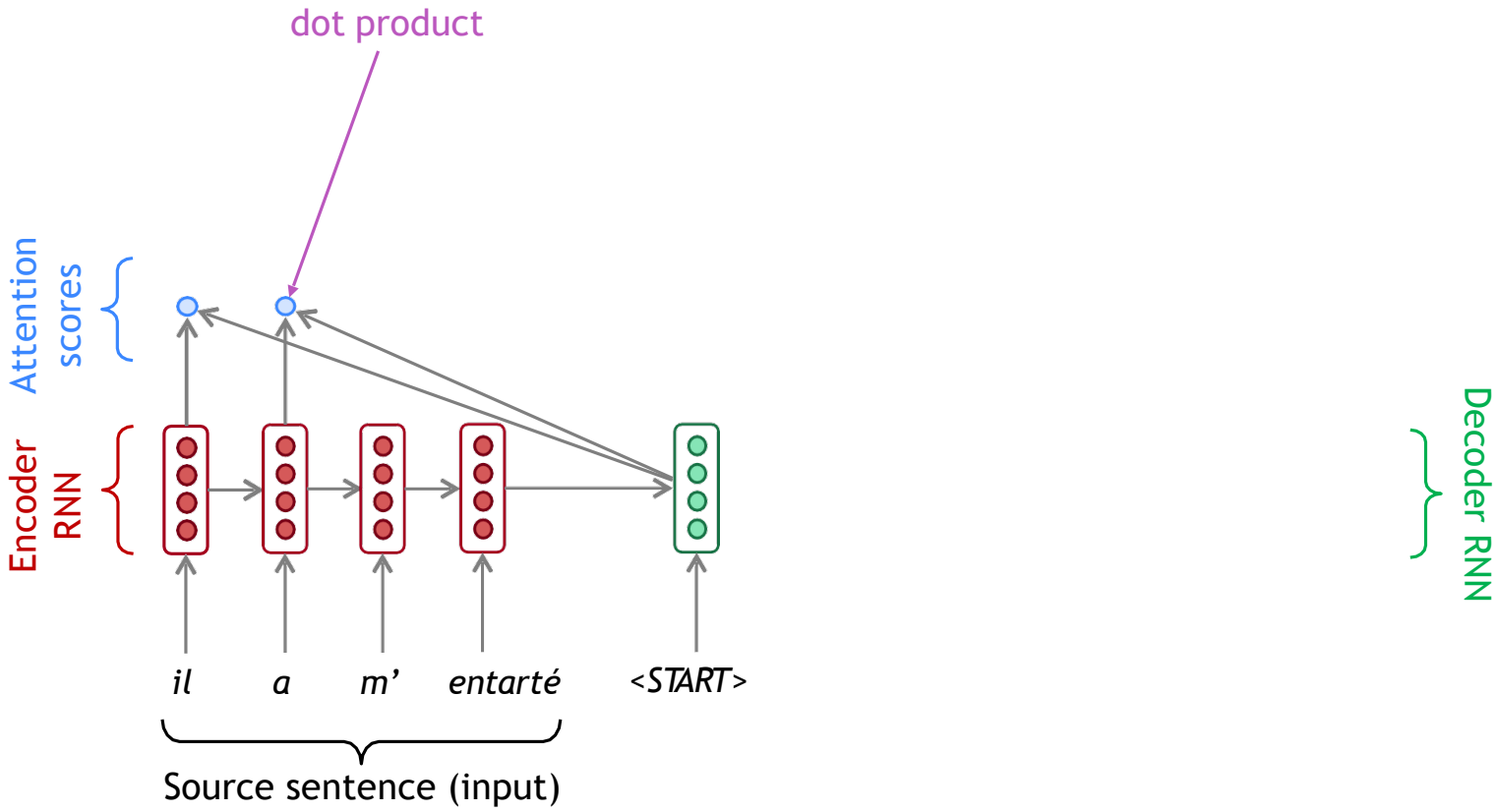
Recap: Sequence-to-sequence



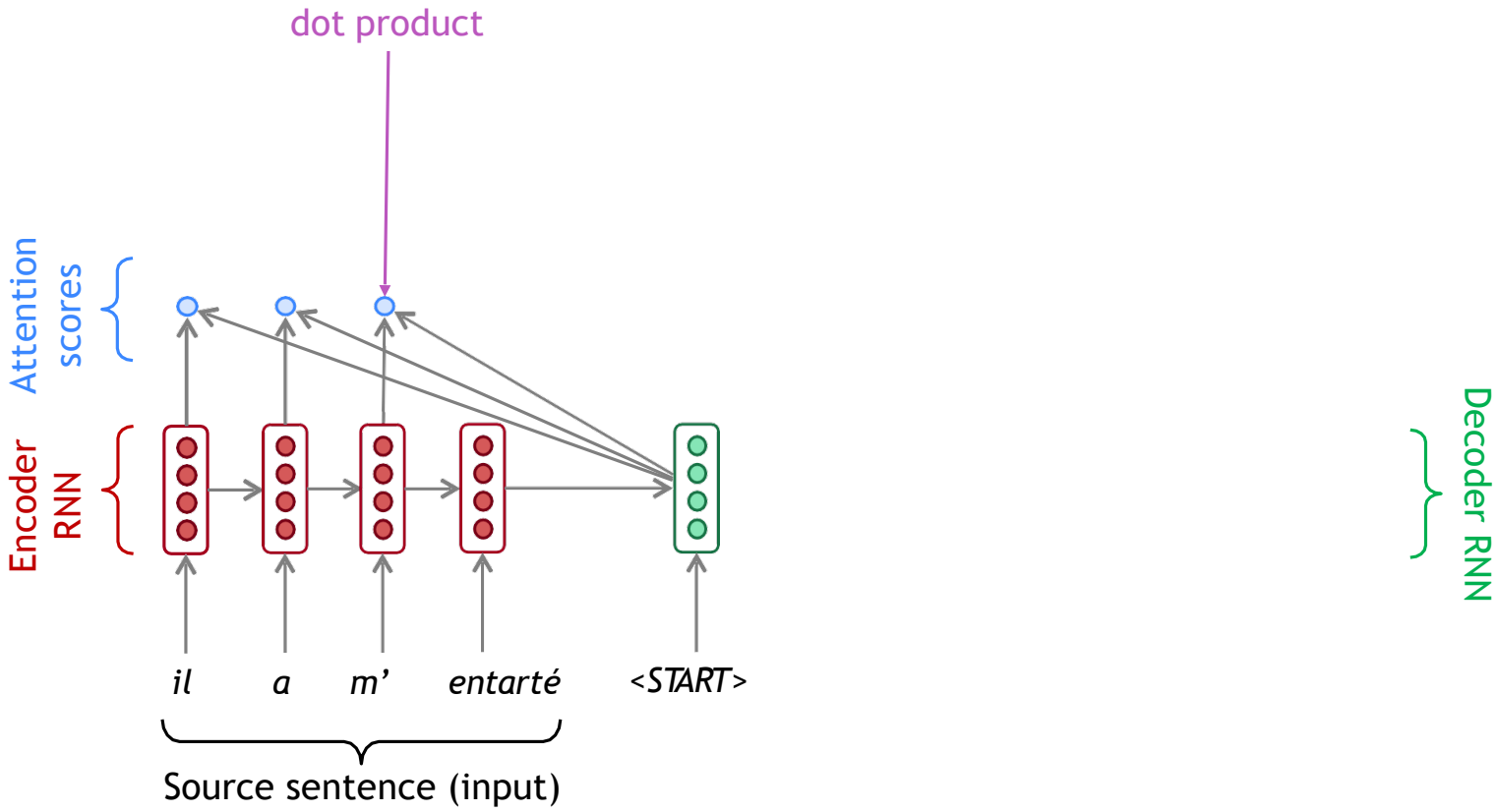
Sequence-to-sequence with attention



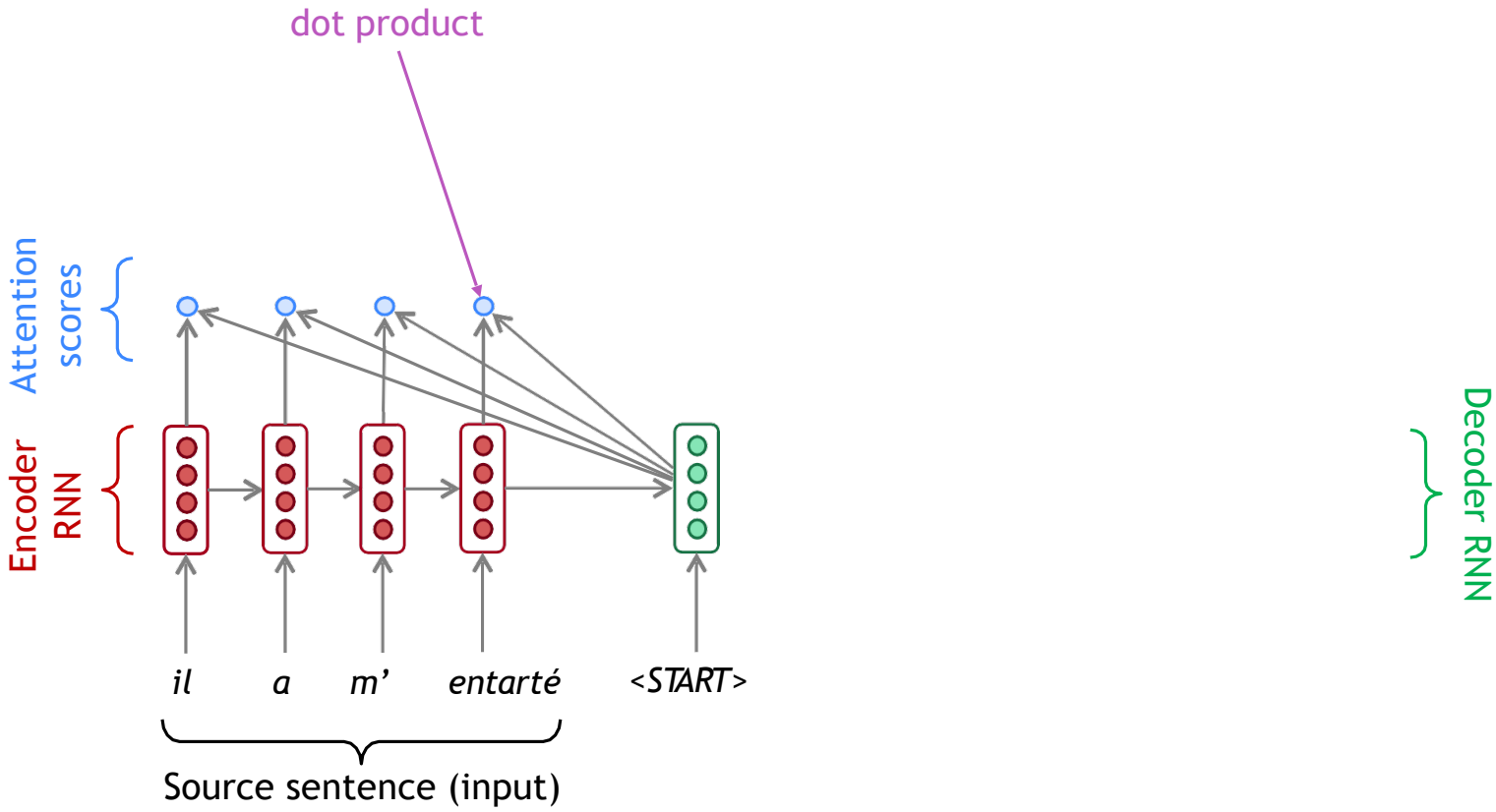
Sequence-to-sequence with attention



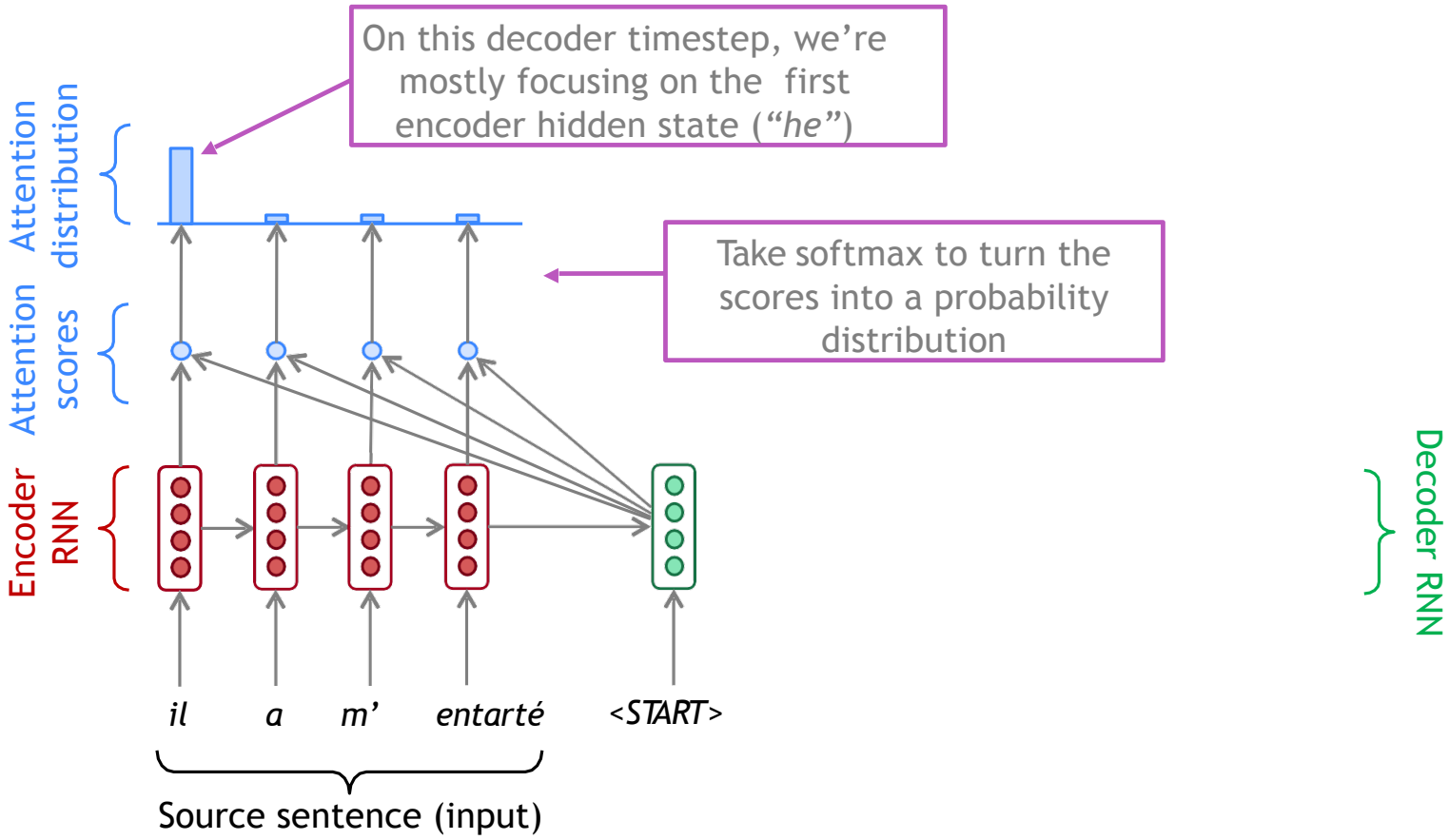
Sequence-to-sequence with attention



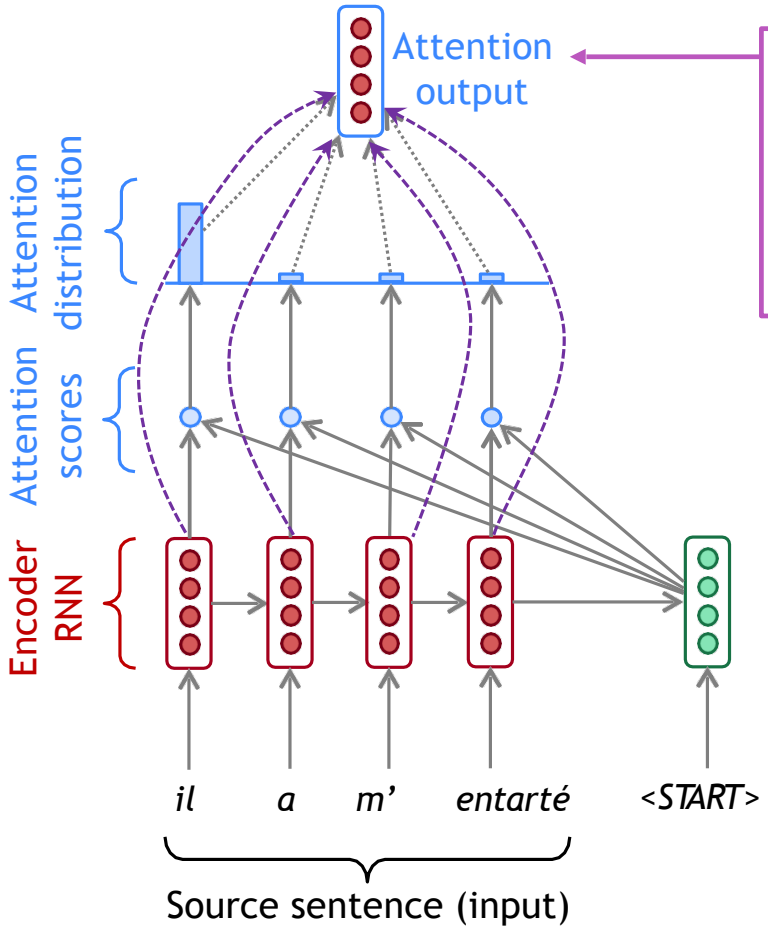
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention

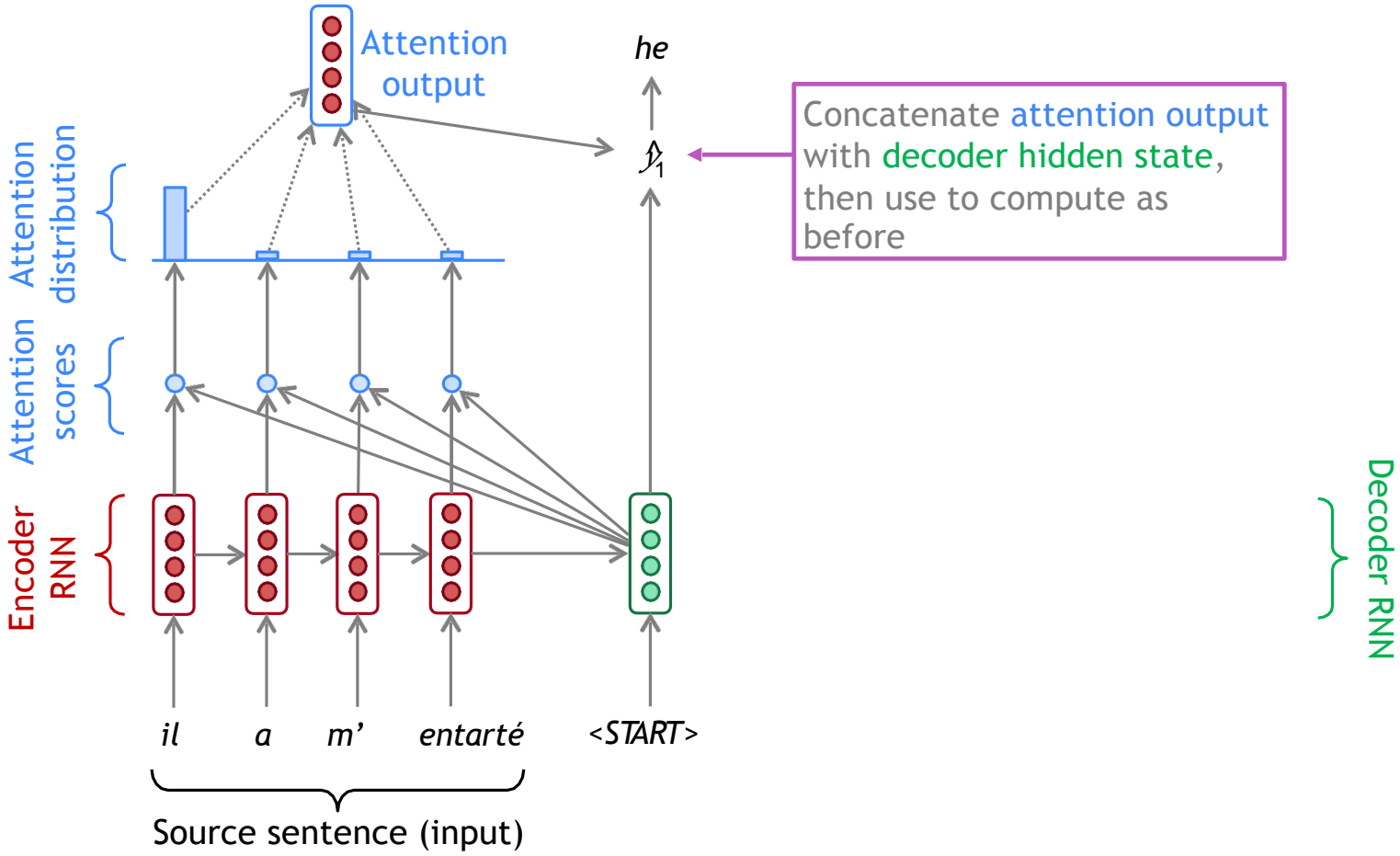


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

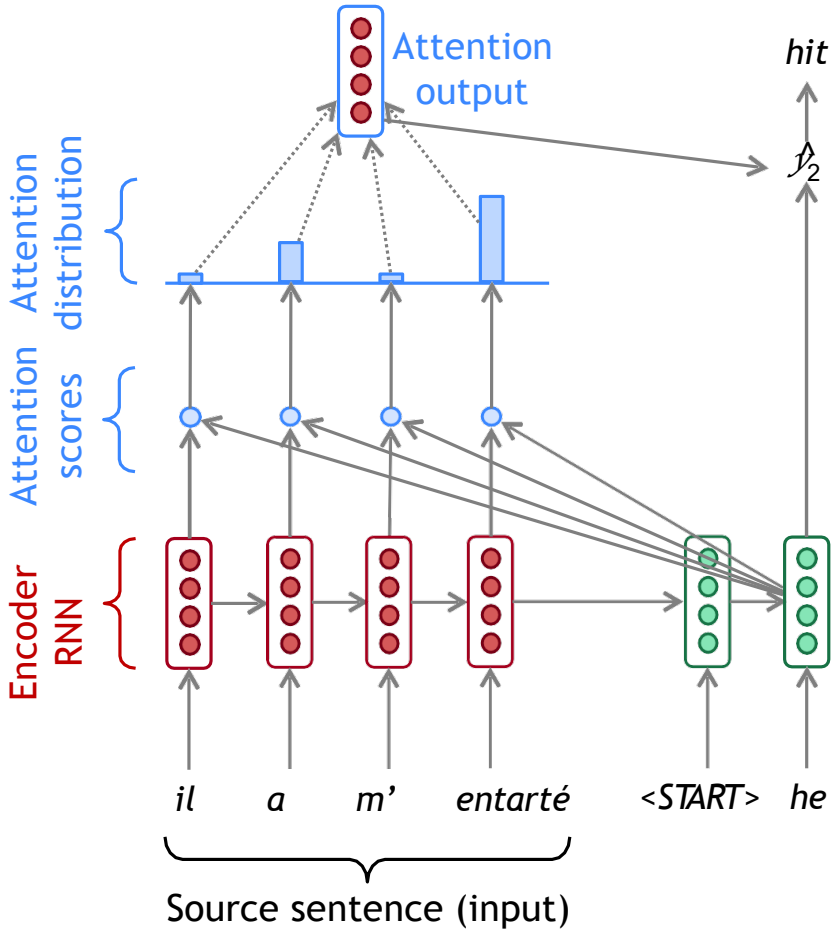
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

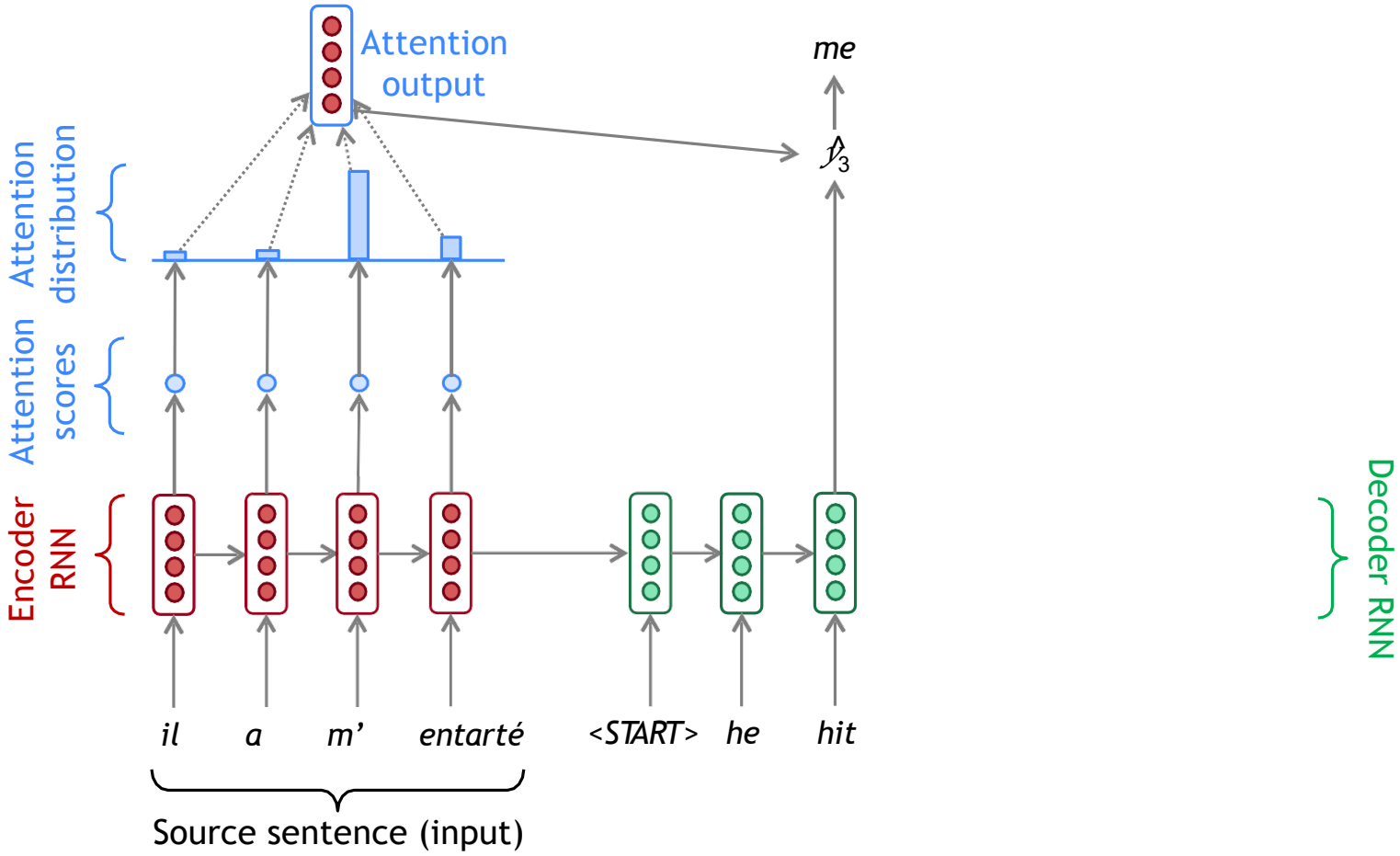
Sequence-to-sequence with attention



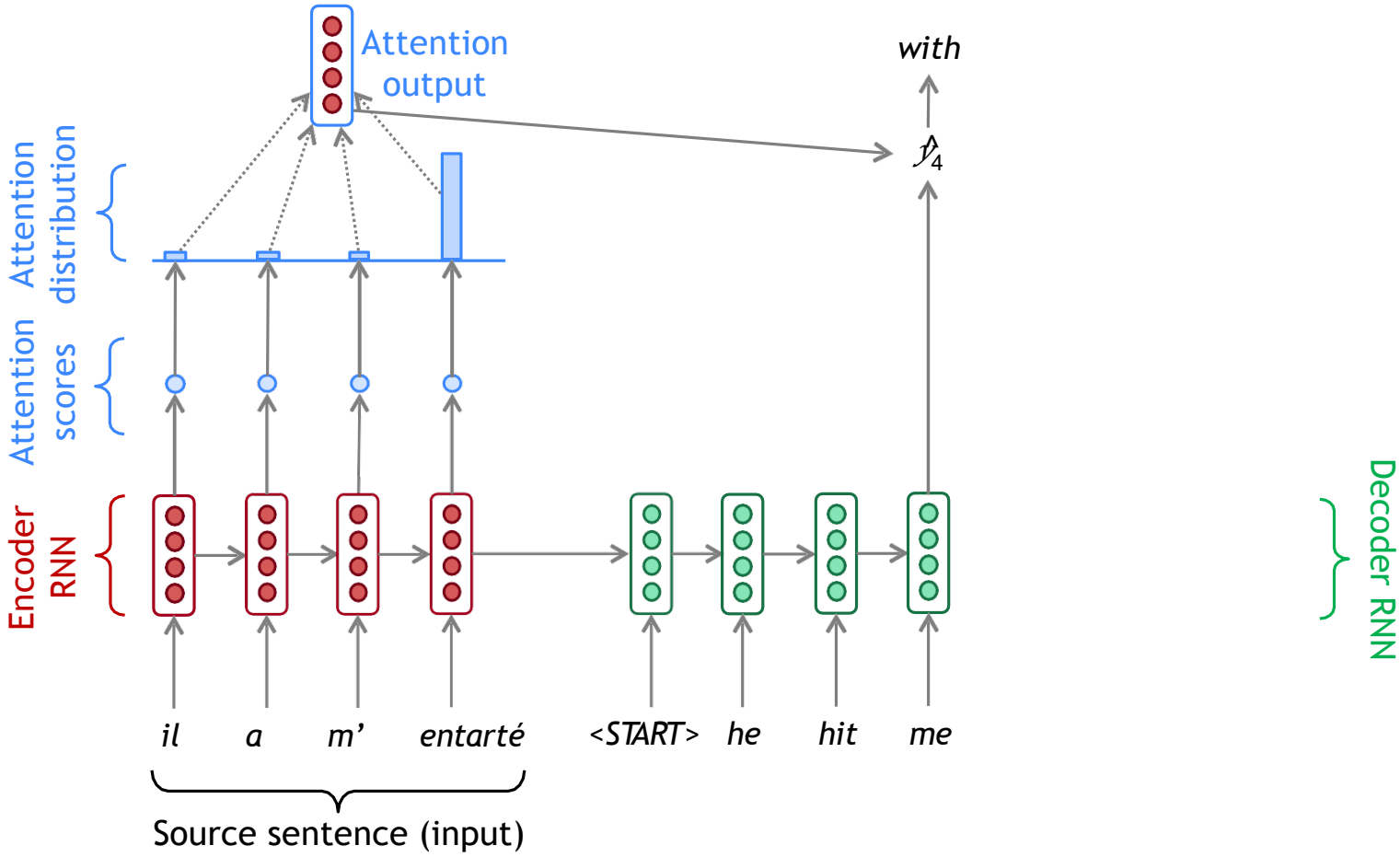
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Keep this in mind:

At the high-level, we are just creating a better representation of a sequence of words.

Autoencoding (Unsupervised Representation Learning)

An autoencoder is a neural network that is trained to attempt to copy its input to its output!

Training the autoencoder to perform the input copying task will result in the last hidden state of encoder taking on useful features which could be used as a feature for downstream tasks (e.g., classification).

Sometimes autoencoding is done as a warm-up for training encoder-decode models.

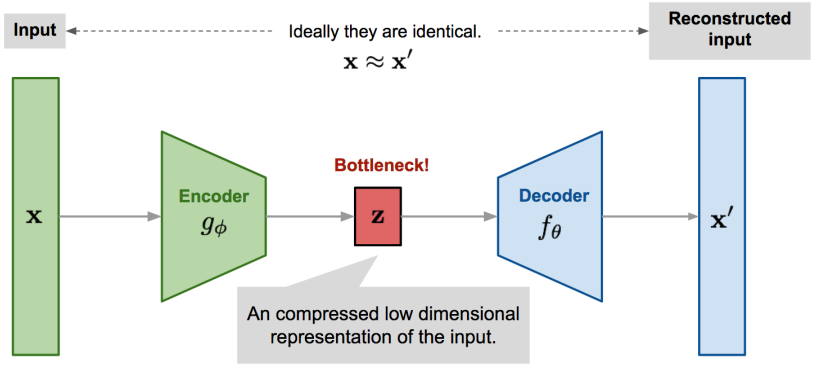
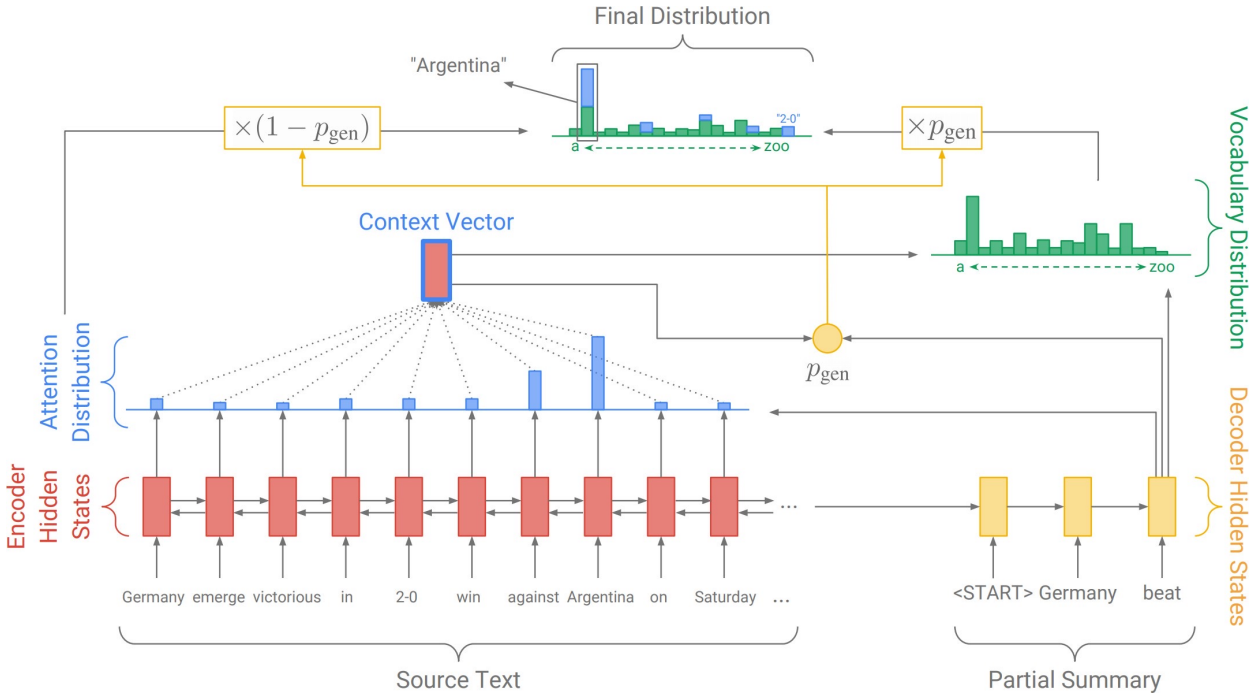


Image from: https://lilianweng.github.io/posts/2018-08-12-vae/

Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). https://www.deeplearningbook.org/contents/autoencoders.html

Text Summarization

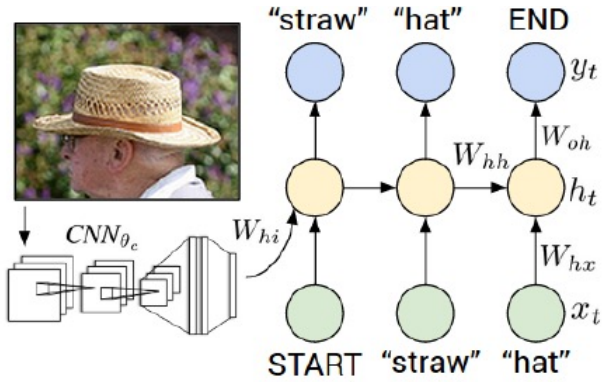


Sketch of the idea: Adjust the Seq2Seq+Attention model by allowing input words to be copied directly to the output.

Get To The Point: Summarization with Pointer-Generator Networks, See et al, 2017 https://arxiv.org/pdf/1704.04368.pdf

Caption Generation

Sketch of the idea: Encode the image using CNN, and use it to initialize the hidden state of RNN. Train just like an RNN-LM.



Generated outputs:

man in black shirt is playing guitar.

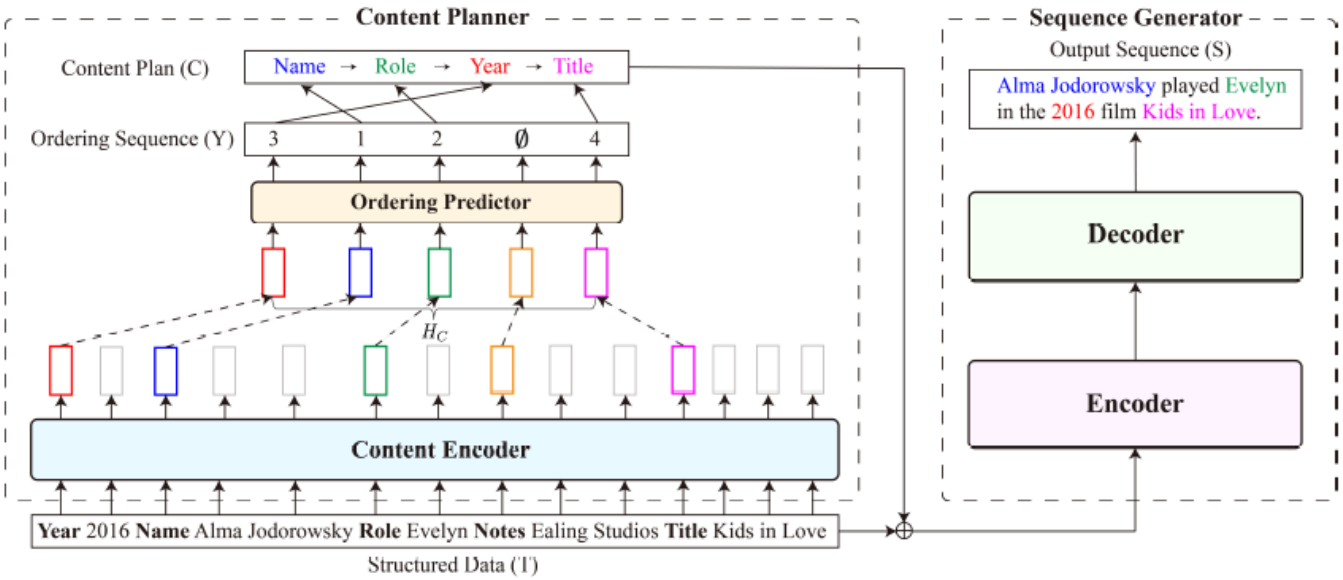
construction worker in orange safety vest is working on road.

two young girls are playing with lego toy.

boy is doing backflip on wakeboard.

Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy et al, 2015 https://arxiv.org/abs/1412.2306

Describing tabular data with text



Sketch of the idea: Encode the table row, then pass the encoding to a seq2seq model to generate the text description.

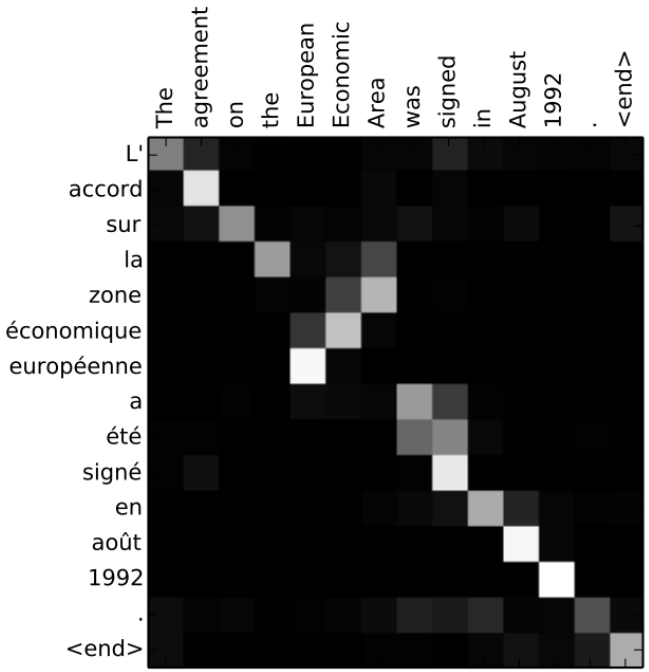
Plan-then-Generate: Controlled Data-to-Text Generation via Planning, Su et al, 2021 https://arxiv.org/abs/2108.13740

Attention for Interpretability

Neural Machine Translation by Jointly Learning to Align and Translate. Bahdanau et al. (2015) introduced the attention mechanism and argued it provides interpretability by showing which input words the model focuses on when generating each output word (i.e. right figure)

Attention is not Explanation. Jain, S. & Wallace, B.C. (2019) challenged the interpretability claims of attention mechanisms. The authors found that attention weights are frequently uncorrelated with gradient-based measures of feature importance, and that very different attention distributions can yield equivalent predictions, concluding that standard attention modules do not provide meaningful explanations.

Attention is not not Explanation: Wiegreffe & Pinter (2019) argued that the claim depends on one's definition of explanation and that attention can still provide meaningful signals when analyzed appropriately with other model elements.



Source: https://jalammar.github.io/images/attention_sentence.png

Interested in the topic of interpretability? You may want to closely follow work by Been Kim and Anthropic.

Overview

1. Recap: Attention mechanism
2. **Attention as a lookup, and self-attention**
3. Transformers
4. Transformer Language Models
5. Pretraining and Finetuning

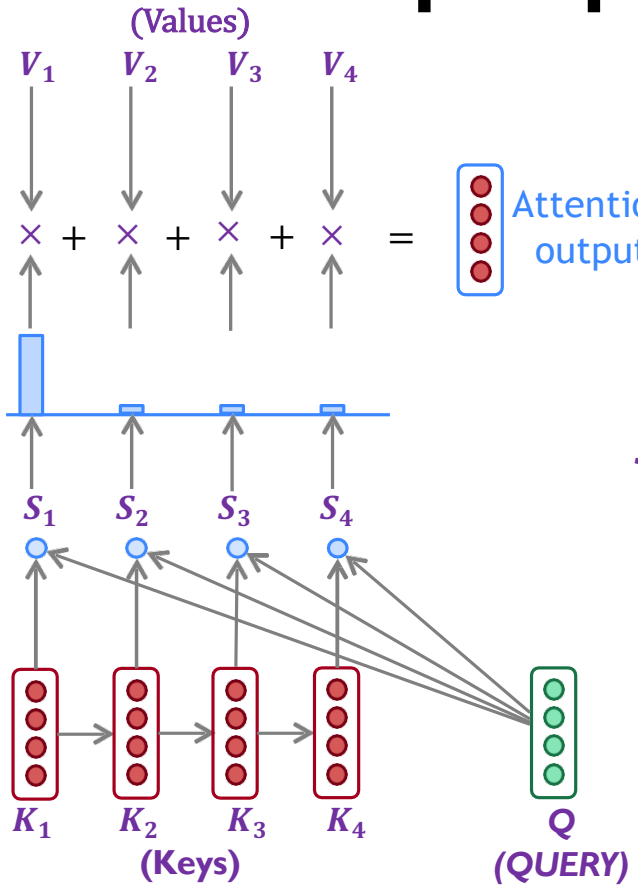
Interpreting attention as a fuzzy database lookup!

- In a database (i.e., hash table) each query matches exactly one key-value pair.
- In attention, each query matches each key to a varying degree.

Attention mimics the retrieval of a **value** V_i for a **query** **Q** based on a K_i in database

$$\text{Attention (Q,V,K)} = \sum_i \text{Similarity(Q, } K_i) \times V_i$$

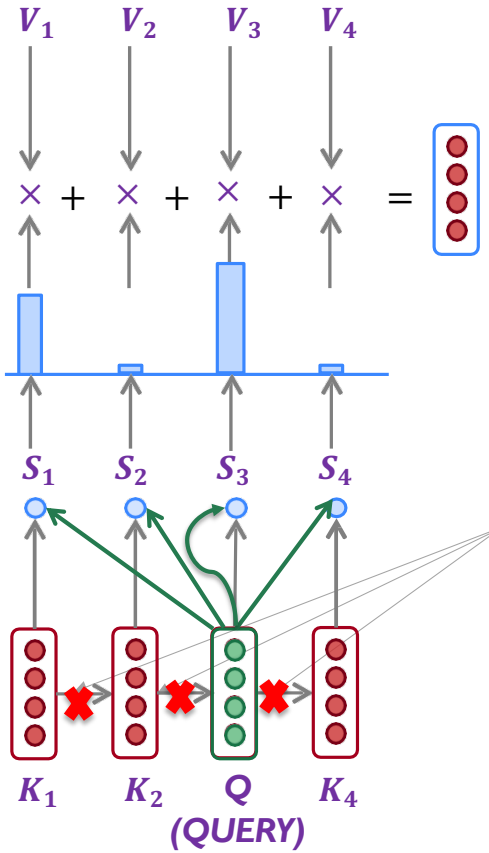
A new notation/perspective



Attention treats each word's representation as a **query** to access and incorporate information from **a set of values.**

$$S_i = f(Q, K_i) = \begin{cases} Q^T K_i & \textit{Dot product} \\ Q^T K_i / \sqrt{d} & \textit{Scaled Dot product} \\ Q^T W K_i & \textit{General Dot product} \end{cases}$$

A new concept – Self-Attention



Self-Attention output: a new representation for Query position

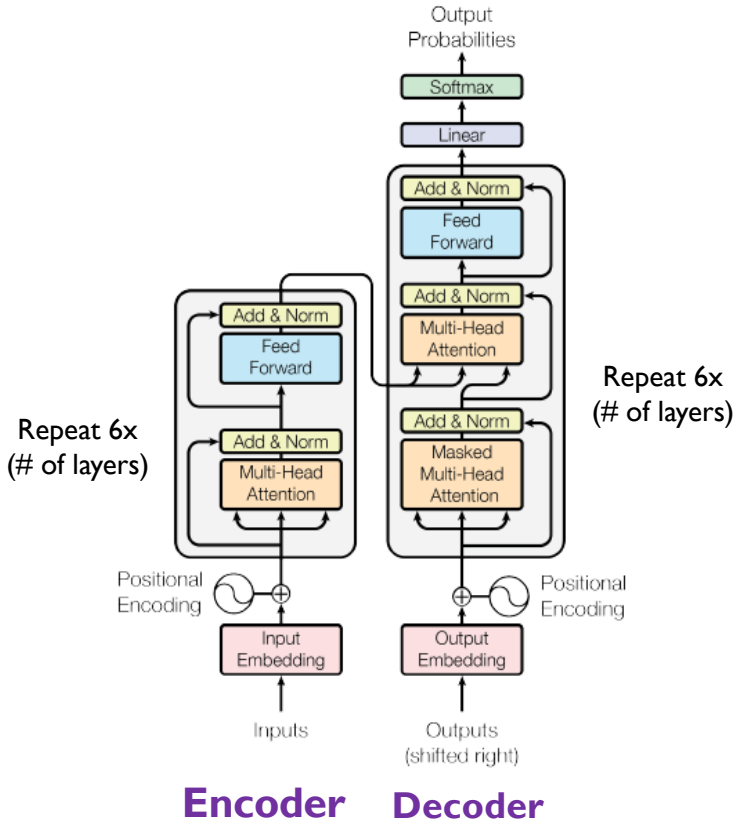
We relied on recurrence in RNNs to learn representations.

Transformer: Just use self-attention instead of recurrence. This way every representation could be computed in parallel!

Overview

1. Recap: Attention mechanism
2. Attention as a lookup, and self-attention
3. **Transformers**
4. Transformer Language Models
5. Pretraining and Finetuning

Attention is all you need! – Vaswani et al., 2017



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

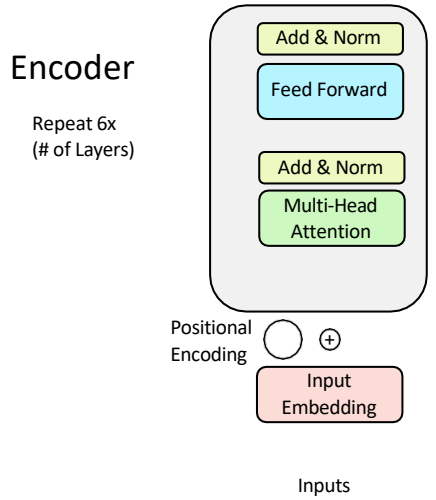
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

A revolution and major shift in processing sequential data!

Transformer Encoder-Decoder

In this section, you will learn exactly how the Transformer architecture works:

- First, we will talk about the Encoder!
- Next, we will go through the Decoder (which is quite similar)!



Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word x_i, calculate its **query**, **key**, and **value**.

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query** and **keys.**

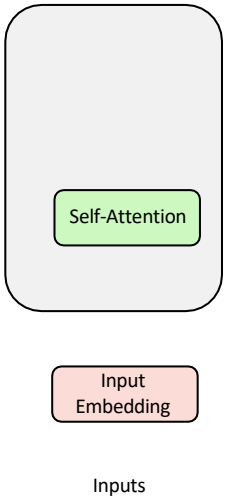
$$e_{ij} = q_i \cdot k_j$$

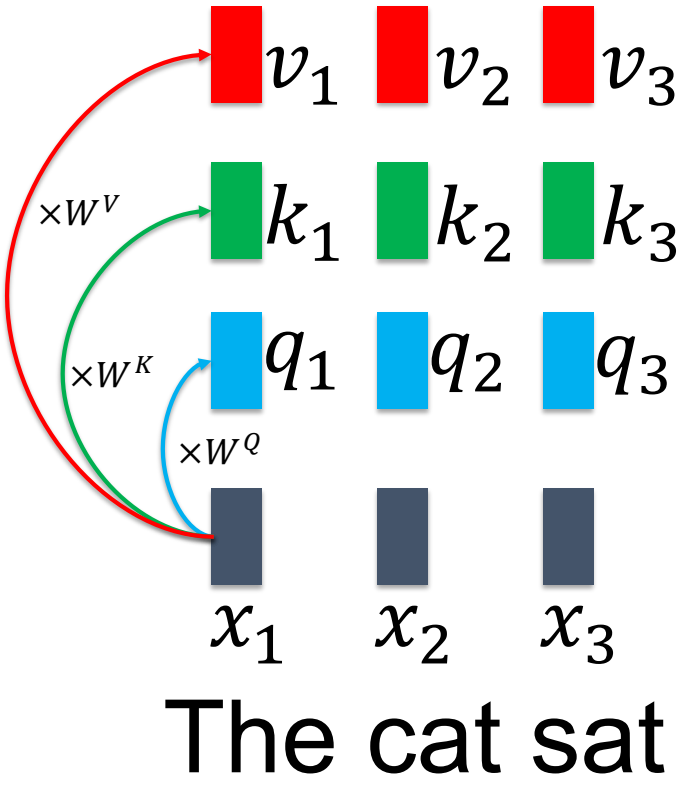
- Step 3: Take the softmax to normalize attention scores.

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

$$Output_i = \sum_j \alpha_{ij} v_j$$





Practice Quiz

Given the sequence “The cat sat” and the following information, which statement is **not** correct?

(a) Query representation of “The” is: [0.05,0.05]
(b) Key representation of “The” is: [0.04,0.08]
(c) Value representation of “The” is: [0.06,0.09]
(d) Key representation of “cat” is: [0.07, 0.16]

$$x_{\text{The}} = [0.1,\ 0.2]$$

$$x_{\text{cat}} = [0.5,\ 0.1]$$

$$x_{\text{sat}} = [0.2,\ 0.8]$$

$$W_Q = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 0.2 & 0.0 \\ 0.1 & 0.4 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.0 & 0.5 \\ 0.3 & 0.2 \end{bmatrix}$$

Sample

Given the sequence “The cat sat” and the following information, what is the self-attention output for query “The”? Use scaled softmax ($\sqrt{2} = 1.414$).

Query rep of The:
Key rep of The:
Key rep of cat:
Key rep of sat:
Value rep of The:
Value rep of cat:
Value rep of sat:

$$W_Q = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 0.2 & 0.0 \\ 0.1 & 0.4 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.0 & 0.5 \\ 0.3 & 0.2 \end{bmatrix}$$

$$x_{\text{The}} = [0.1,\ 0.2]$$

$$x_{\text{cat}} = [0.5,\ 0.1]$$

$$x_{\text{sat}} = [0.2,\ 0.8]$$

Sample

Given the sequence “The cat sat” and the following information, what is the self-attention output for query “The”? Use scaled softmax ($\sqrt{2} = 1.414$).

Query rep of The: [0.05,0.05]
Key rep of The=[0.04,0.08]
Key rep of cat=[0.11,0.04]
Key rep of sat=[0.12,0.32]
Value rep of The=[0.06,0.09]
Value rep of cat= [0.03,0.27]
Value rep of sat=[0.24,0.26]

$$W_Q = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 0.2 & 0.0 \\ 0.1 & 0.4 \end{bmatrix}, \quad W_V = \begin{bmatrix} 0.0 & 0.5 \\ 0.3 & 0.2 \end{bmatrix}$$

$$x_{\text{The}} = [0.1,\ 0.2]$$

$$x_{\text{cat}} = [0.5,\ 0.1]$$

$$x_{\text{sat}} = [0.2,\ 0.8]$$

Attention scores:
The,The: [0.05,0.05] x [0.04,0.08] = 0.05×0.04+0.05×0.08=0.006
The, cat: [0.05,0.05] x [0.11,0.04] = 0.05×0.11+0.05×0.04=0.0075
The, sat: [0.05,0.05] x [0.12,0.32] = 0.05×0.12+0.05×0.32=0.022

Scaled scores (scaling factor = 1.414)
The,The (scaled): 0.006/1.414 = 0.006
The, cat (scaled): 0.0075/1.414 = 0.005
The, sat (scaled): 0.022/1.414 = 0.016

Attention distribution (softmax over scaled scores): [0.332, 0.332, 0.336]

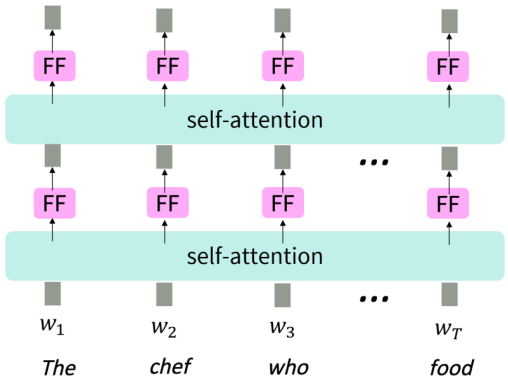
Attention output: 0.332x[0.06,0.09] + 0.332x[0.03,0.27] + 0.336x[0.24,0.26] = [0.1105, 0.20688]

But Self-Attention is not enough!

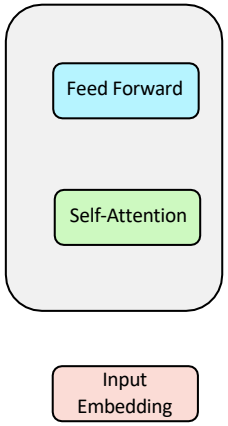
- **Problem:** Since there are no element-wise non-linearities, self- attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

Equation for Feed Forward Layer

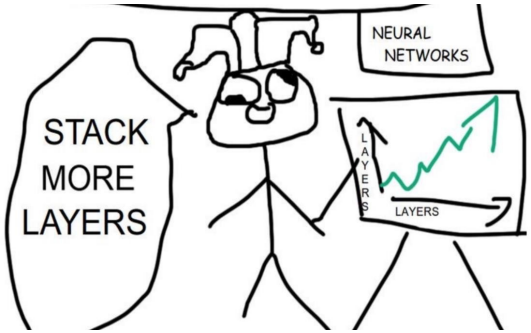
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$

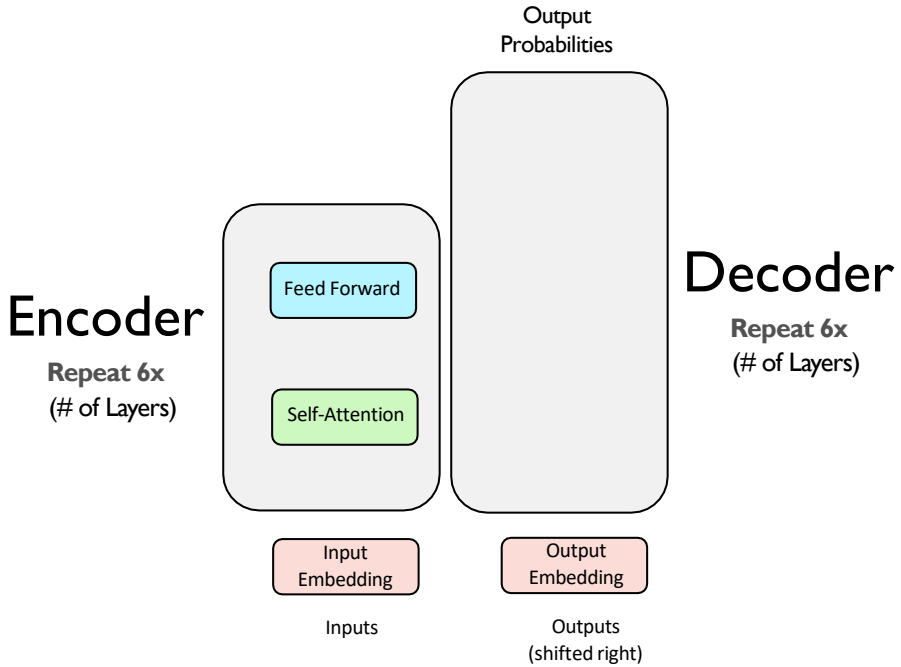


Encoder



A few more tricks!





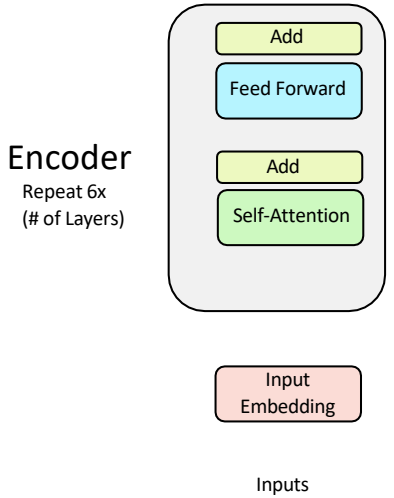
Stack up more layers, and apply

- Training Trick #1: Residual Connections
- Training Trick #2: Layer Normalization
- Training Trick #3: Scaled Dot Product Attention

Training Trick #1: Residual Connections

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



[He et al., 2016]

Training Trick #2: Layer Norm

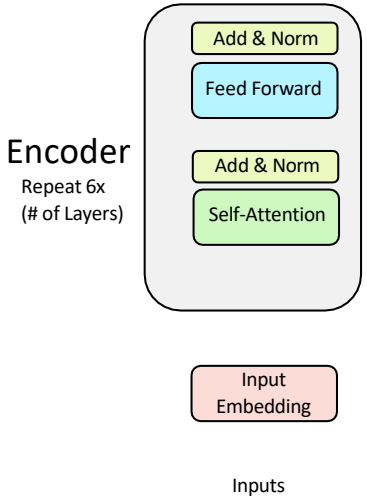
- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.

- **Solution:** Reduce variation by **normalizing** to 0 mean and standard deviation of 1 within each **layer**.

Mean: $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$ Standard Deviation: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$

H denotes the number of hidden units in a layer

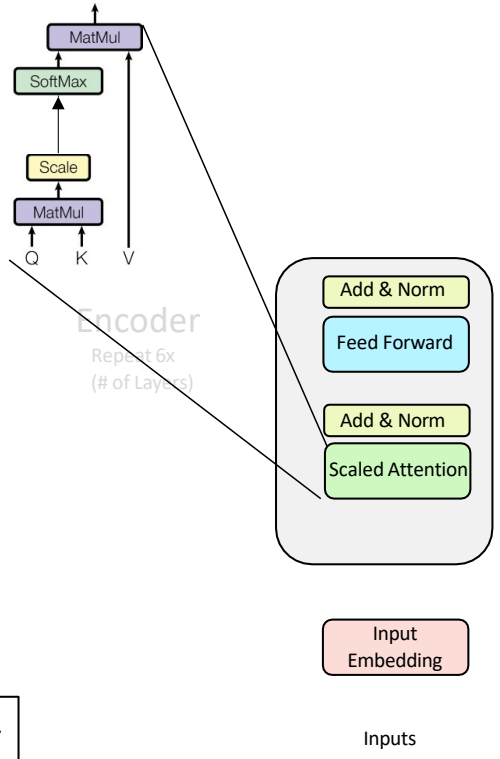
$$x^{\ell\prime} = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$



[Ba et al., 2016]

Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and variance of vector elements is 0 and 1, respectively.
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k



Quick Statistics Review:

If q and k were both mean 0 and variance 1. Then their product will also have mean 0, but variance d_k. So division by d_k keeps the variance of the product also at 1.

Updated Self-Attention Equation:

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

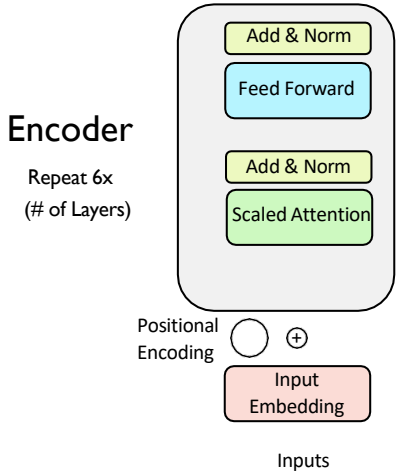
Almost done with the encoder!

- We have a major problem! Order doesn't impact the network at all!
- This seems wrong!
- Solution: Inject Order Information through Positional Encodings!
  - Consider representing each **sequence index** as a **vector**

    $p_i \in \mathbb{R}^d$, for $i \in \{1,2,\dots,T\}$ are position vectors
  - Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!

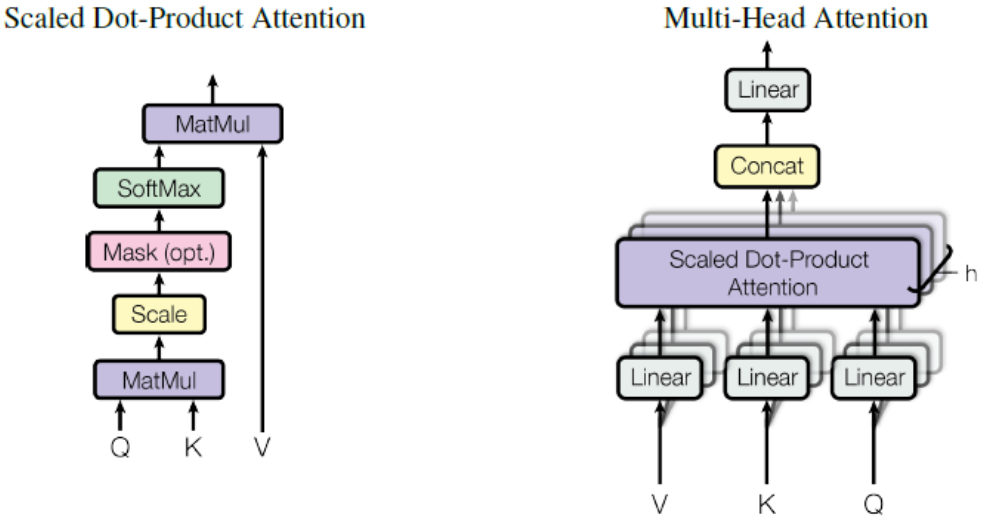
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

[For more details on p_i read the paper.]



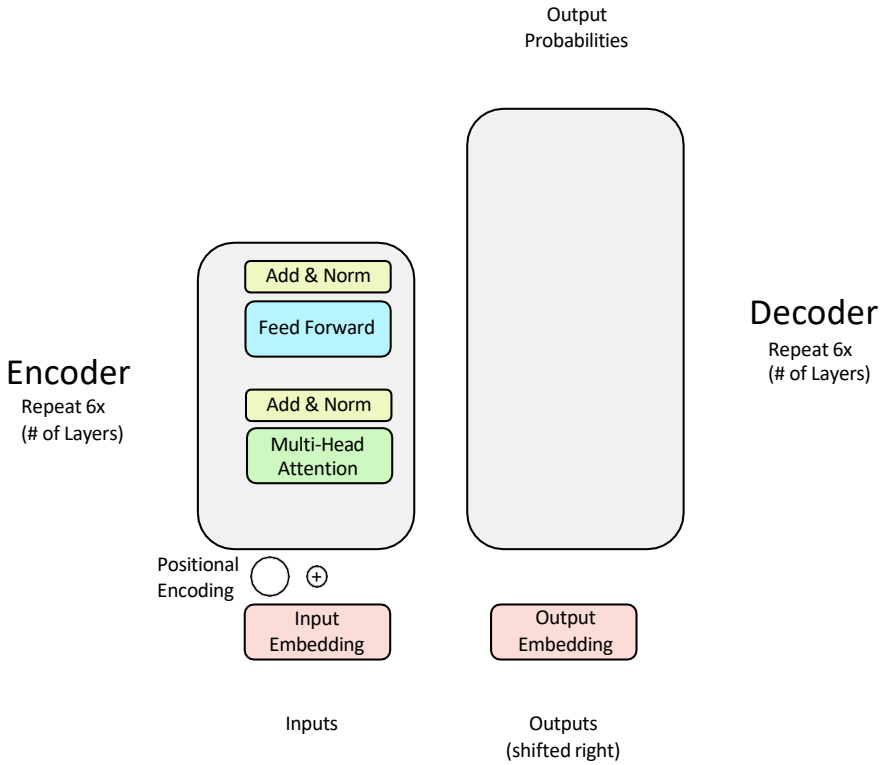
Multi-Head Self-Attention: k heads are better than 1!

- **High-Level Idea:** Let's perform self-attention multiple times in parallel and combine the results.
- **Multiple attention "heads"** through multiple Q,K,V matrices
- Each attention head performs attention independently.



[Vaswani et al., 2017]

Finished with Encoder, now let's move to Decoder!



Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?

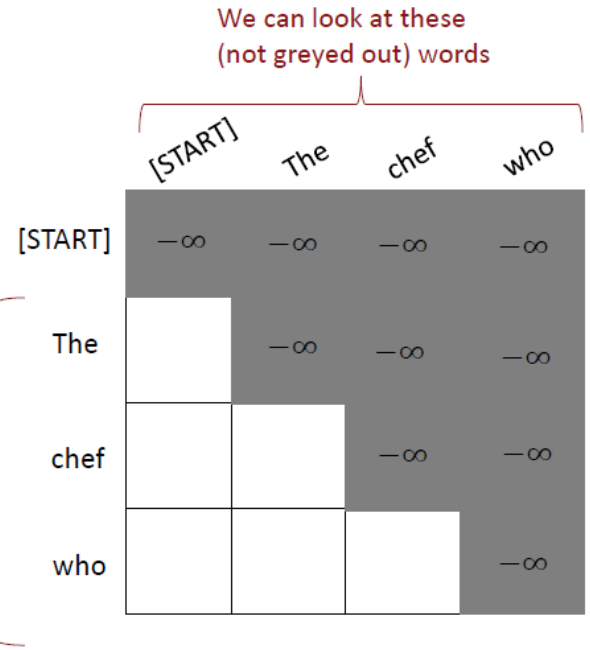
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.

Masking the Future in Self-Attention

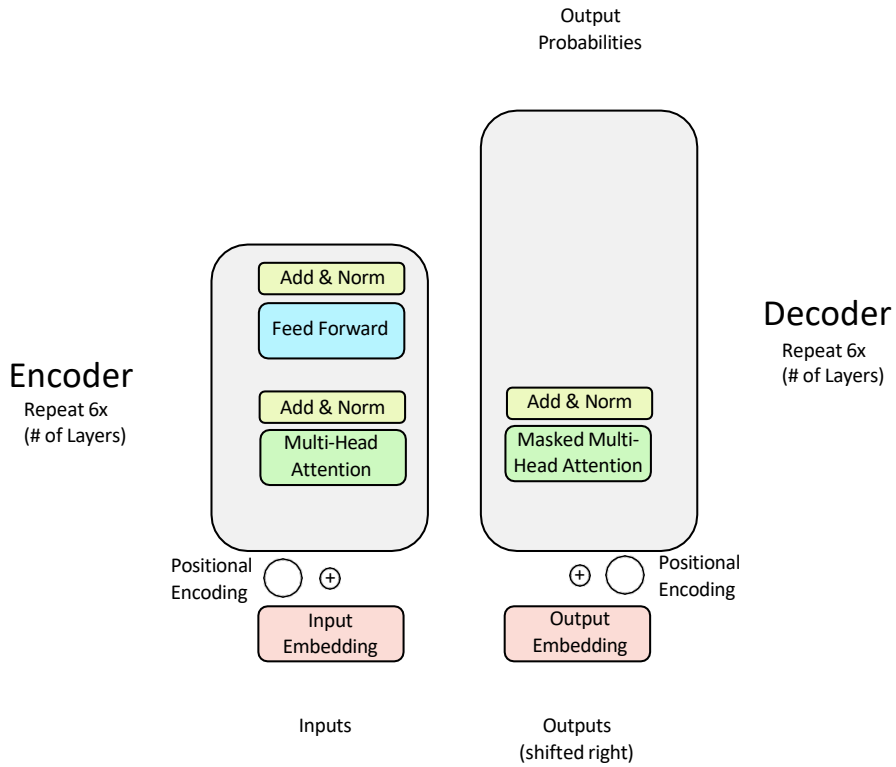
- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$

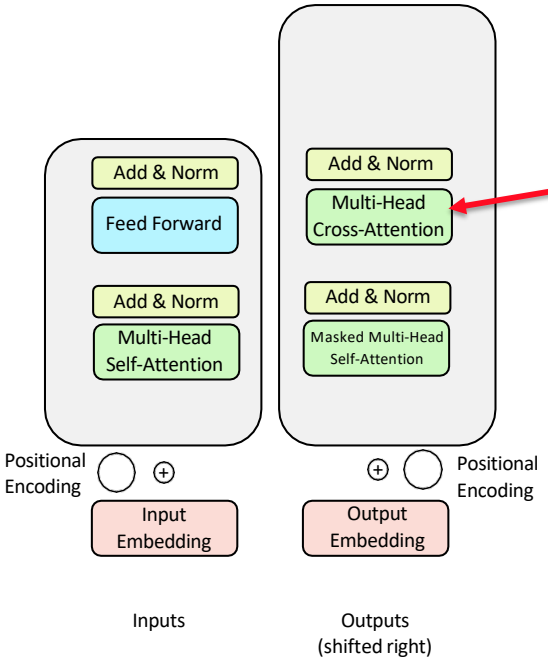
Question: Why this does what we want?



Decoder: Masked Multi-Head Self-Attention



Encoder-Decoder Attention

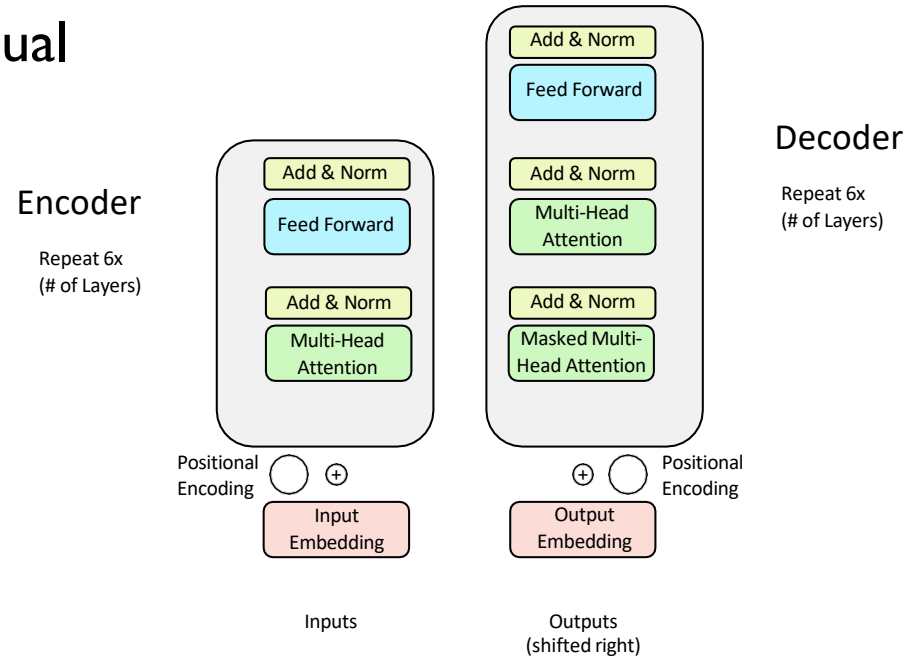


For an encoder-decoder design, we need **cross-attention** when, where

- Keys (K) and values (V) are drawn from the **encoder**
- **and** queries (Q) are from **decoder**

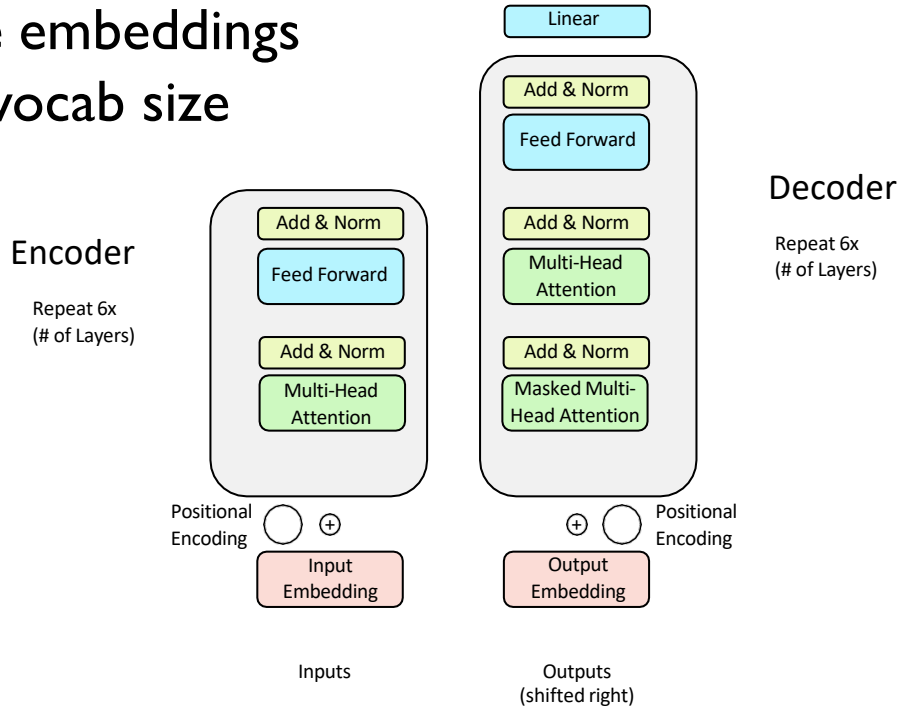
Finishing the decoder …

- Add a feed forward layer (with residual connections and layer norm)



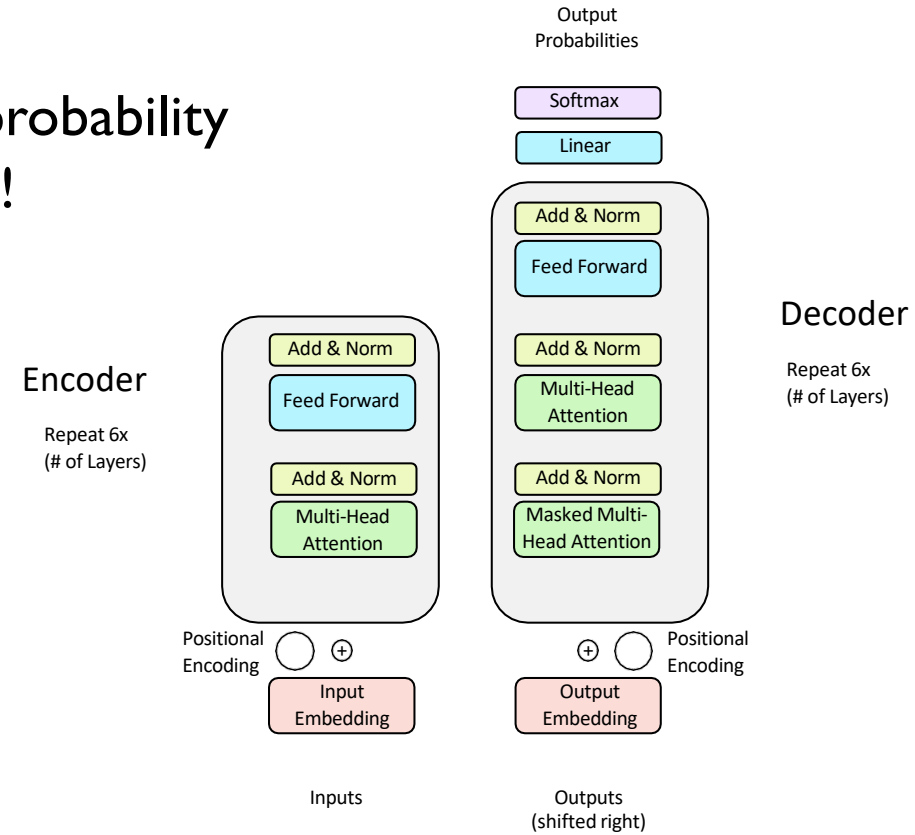
Finishing the decoder …

- Add a final linear layer to project the embeddings into a much longer vector of length vocab size



Finishing the decoder …

- Add a final softmax to generate a probability distribution of possible next words!

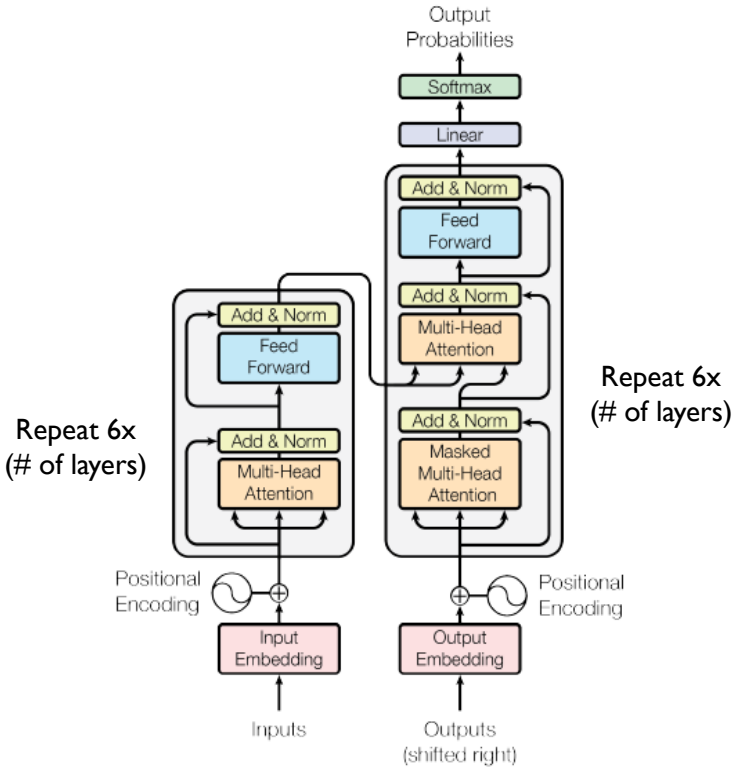


Recap of the Transformer Architecture

The paper proposing Transformers, presented them for an Encoder-Decoder architecture.

But Transformers are just another type of neural networks, and there are Encoder-only and Decoder-only architectures that use Transformers.

Today's LLMs are mostly decoder-only.



https://arxiv.org/abs/1706.03762

Overview

1. Recap: Attention mechanism
2. Attention as a lookup, and self-attention
3. Transformers
4. **Transformer Language Models**
5. Pretraining and Finetuning

Pretrained Language Models

The neural architecture influences the type of pretraining, and natural use cases.

Encoder-Decoders

- What's the best way to pretrain them?
- **Examples:** T5, BART, FLAN-T5

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** ChatGPT, Gemini, Claude, LLaMA, Qwen

Encoders

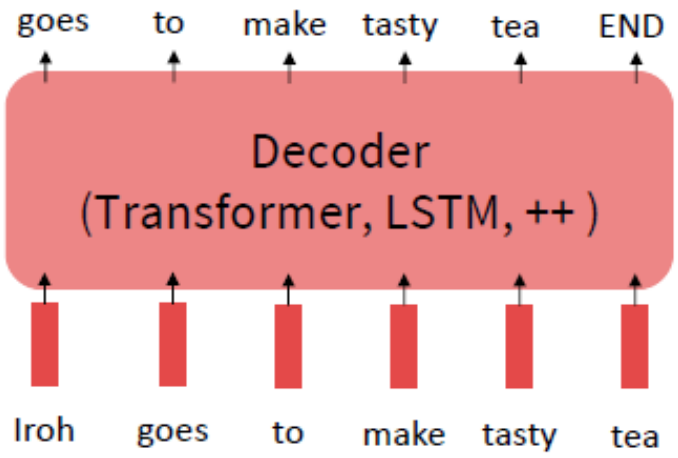
- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa

Pretraining and Finetuning paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

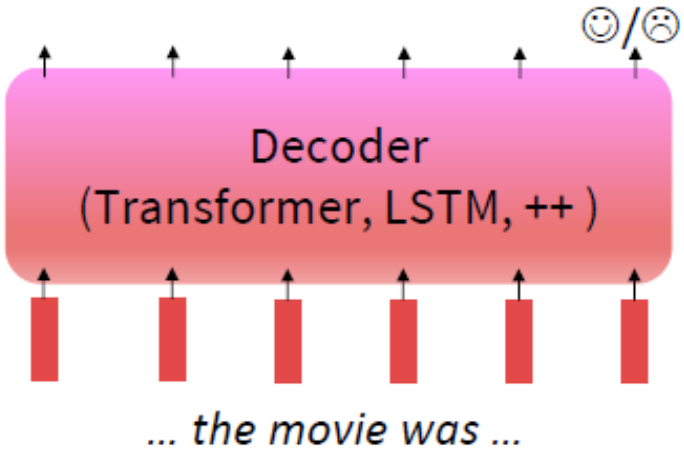
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



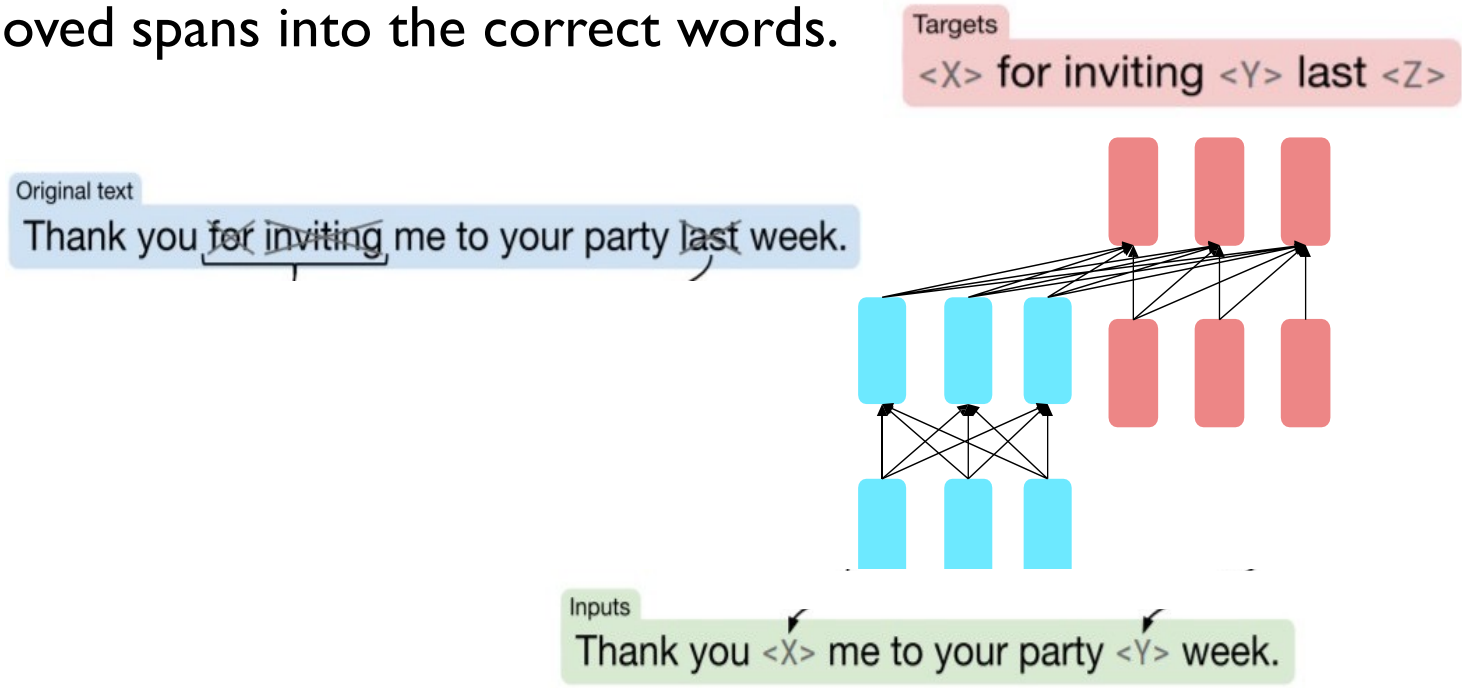
Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Pre-training: T5 Model (Encoder-Decoder)

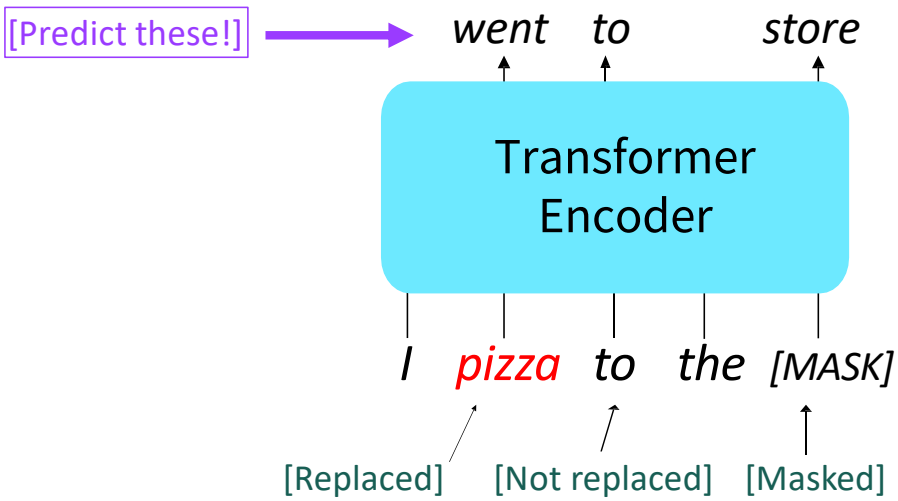
T5 (Raffel et al., 2020) pretraining replace different-length spans from the input with unique placeholders; then trains the model to decode the removed spans into the correct words.



Pre-training: BERT Model (Encoder)

- BERT Devlin et al., (2019):

  Take a sentence from the training data (I went to the store) and pretrain to predict a random 15% of input tokens where:



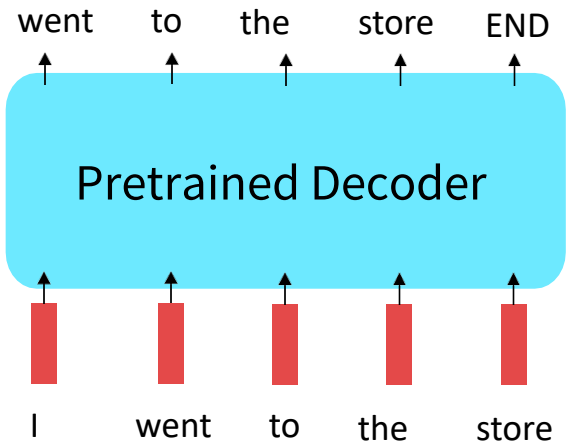
- Replace input word with [MASK] 80% of the time
- Replace input word with a random token 10% of the time
- Leave input word unchanged 10% of the time (but still predict it!)

	Heads	# Param	Layers	Hidden state dimension
Base	12	110 M	12	768
Large	16	240 M	24	1024

Pre-training: GPT Models (Decoder)

Generative Pretrained Transformer (Radford et al, 2018) - essentially most LLMs we know these days are pre-trained in a similar way - is a decoder-only transformer trained on predicting next word based on previous words by computing $P(x_t|x_{1\ldots t-1})$

- Take a sentence from the training data (I went to the store) and pretrain on the language modelling (i.e., predicting next word) task:

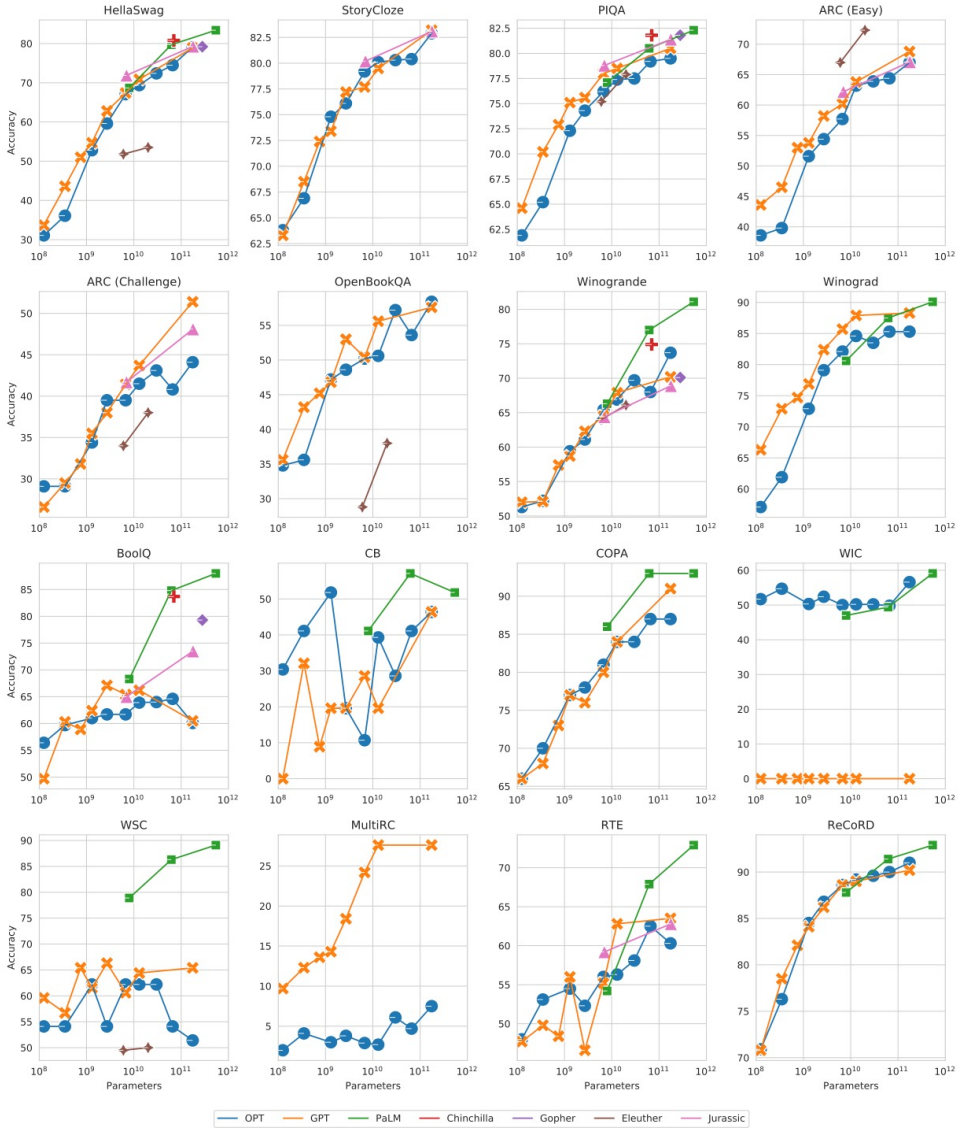


	Heads	# Param	Layers	Hidden state dimension
GPT	12	117 M	12	768
GPT2	12	1542 M	48	1600
GPT3	128	175 B	96	12288

Performance scales with model size

Comparison of different decoders and model sizes in the zero-shot setting for 16 tasks

OPT: Open Pre-trained Transformer Language Models (2022)



Scaling Law paper (a must read)

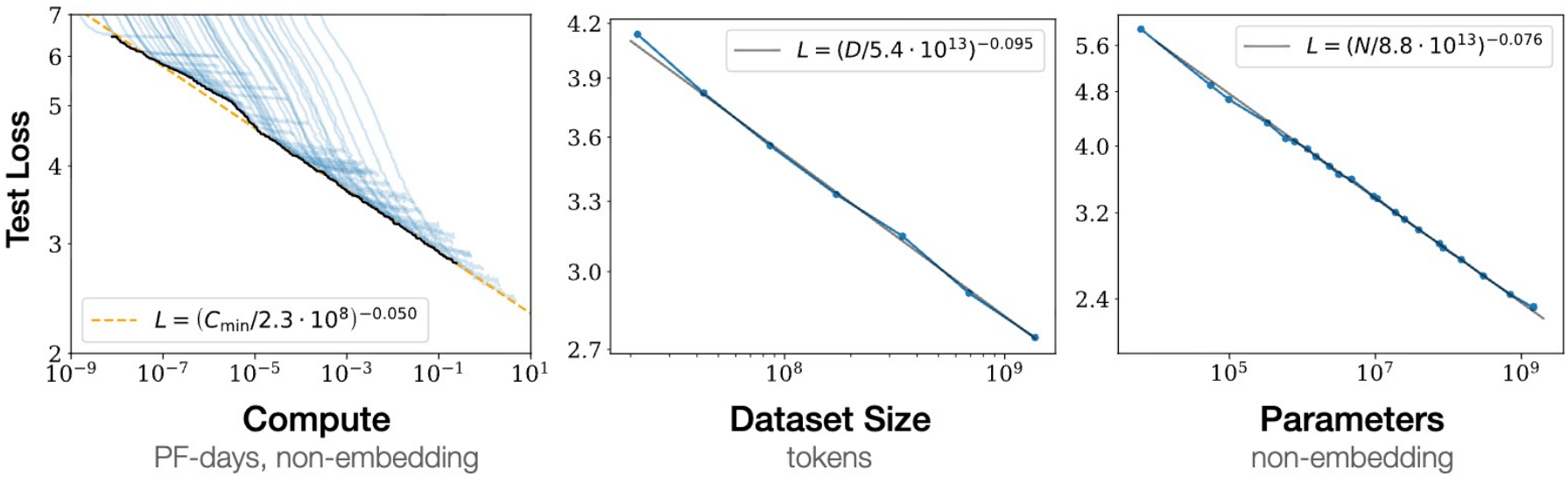
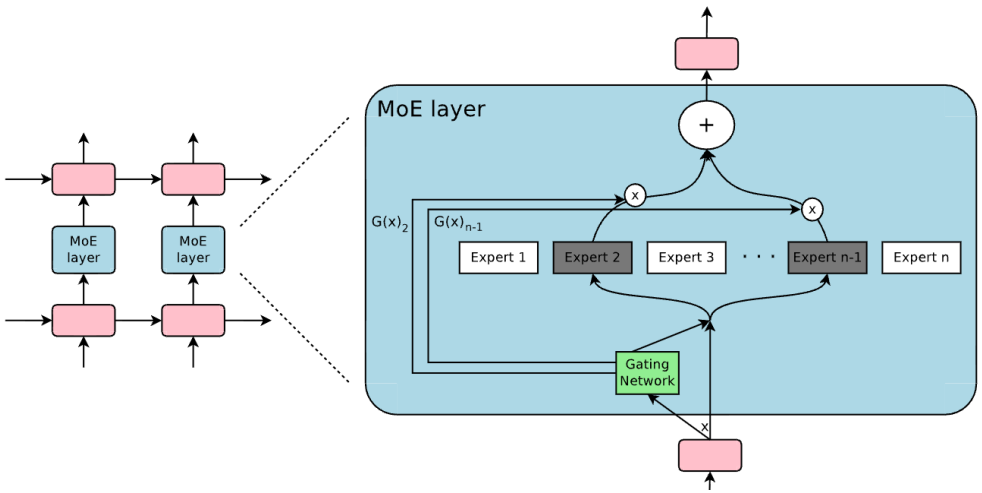


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Scaling Laws for Neural Language Models (Kaplan et al., 2020) https://arxiv.org/pdf/2001.08361

Mixture-of-Experts

You can trace this idea back to Hinton's 2017 work on RNNs

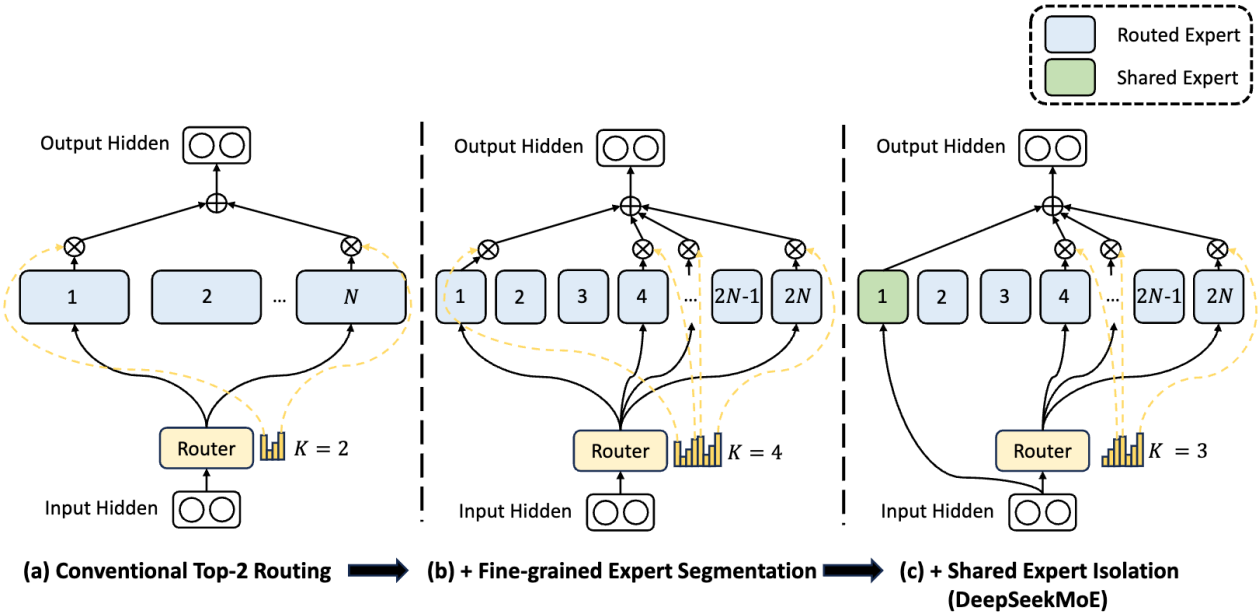


From abstract:

"The capacity of a neural network to absorb information is limited by its number of parameters. … In this work, we finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters."

The implementation of this idea in Transformers FFN: Instead of having a single feed-forward network within the feed-forward component of the transformer block, we create several feed-forward networks, *each with their own independent weights*. We call these experts. For each token we must choose k experts to process it. A simple routing method linearly projects the token to an N-dimensional score vector (i.e., where N is the total number of experts), and applies softmax to get a distribution, and routes the token to the top-K experts.

Mixture-of-Experts meeting LLMs' FFN



DeepSeekMoE
https://arxiv.org/pdf/2401.06066

	Qwen2.5-72B Base	Qwen2.5-Plus Base	LLaMA-4-Maverick Base	DeepSeek-V3 Base	Qwen3-235B-A22B Base
# Architecture	Dense	MoE	MoE	MoE	MoE
# Total Params	72B	271B	402B	671B	235B
# Activated Params	72B	37B	17B	37B	22B

https://qwenlm.github.io/blog/qwen3

Overview

1. Recap: Attention mechanism
2. Attention as a lookup, and self-attention
3. Transformers
4. Transformer Language Models
5. **Pretraining and Finetuning**

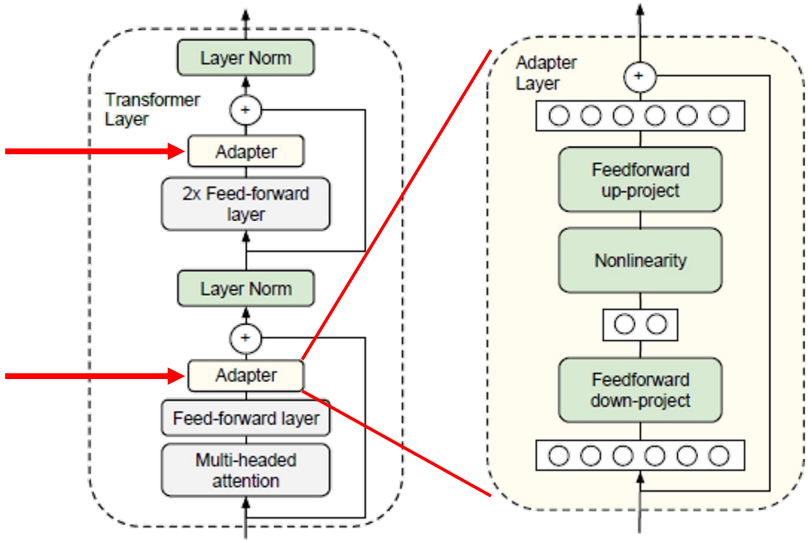
Full Finetuning vs. Parameter-Efficient Finetuning

Full Finetuning every parameter in a pretrained small language model works well, but is compute-intensive.

Lightweight finetuning methods adapt pretrained models in a constrained way which leads to less overfitting and/or more efficient finetuning and inference.

This is generally referred to as Parameter-Efficient Fine-Tuning (PEFT).

Adaptors



	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_mm	QNLI	RTE	Total
BERT_LARGE	9.0×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	1.3×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	1.2×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

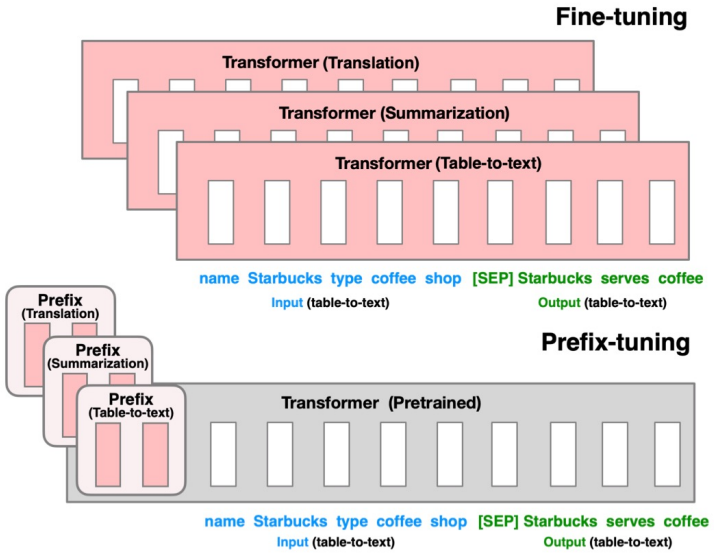
Table 1. Results on GLUE test sets scored using the GLUE evaluation server. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. Adapter tuning achieves comparable overall score (80.0) to full fine-tuning (80.4) using 1.3× parameters in total, compared to 9×. Fixing the adapter size to 64 leads to a slightly decreased overall score of 79.6 and slightly smaller model.

Parameter-Efficient Transfer Learning for NLP, Houlsby et al., 2019

Prefix-Tuning

	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2$_{\text{MEDIUM}}$																				
FT-FULL	68.8	8.71	46.1	71.1	2.43	**64.7**	26.7	45.7	**0.46**	0.30	0.38	**0.33**	0.78	0.54	46.2	**0.39**	**0.46**	**0.50**	**0.94**	**0.39**
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	**2.47**	60.5	**47.9**	54.8	0.43	**0.38**	**0.41**	0.35	**0.46**	**0.39**	45.2	0.38	**0.46**	**0.50**	**0.94**	**0.39**
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	**0.94**	0.33
PREFIX(0.1%)	**70.3**	**8.82**	**46.3**	**72.1**	2.46	62.9	45.3	**55.0**	0.44	0.37	**0.41**	0.35	0.51	0.42	**46.4**	0.38	**0.46**	**0.50**	**0.94**	**0.39**
GPT-2$_{\text{LARGE}}$																				
FT-FULL	68.5	8.78	46.0	69.9	2.45	**65.3**	43.1	55.5	**0.46**	0.38	**0.42**	**0.33**	0.53	0.42	**47.0**	**0.39**	0.46	**0.51**	**0.94**	**0.40**
Prefix	**70.3**	**8.85**	**46.2**	**71.7**	**2.47**	63.4	**47.7**	**56.3**	0.45	**0.39**	**0.42**	0.34	**0.48**	**0.40**	46.7	**0.39**	**0.45**	**0.51**	**0.94**	**0.40**
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

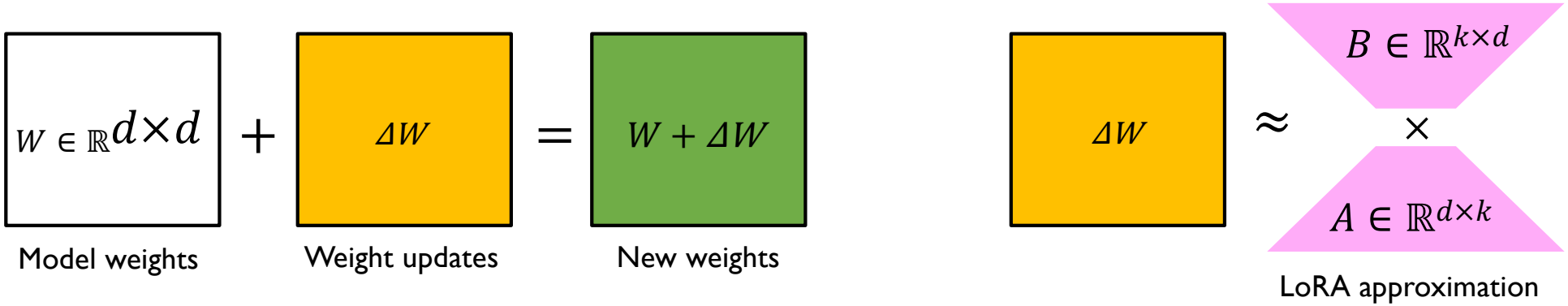
Table 2: Metrics (higher is better, except for TER) for table-to-text generation on E2E (left), WebNLG (middle) and DART (right). With only 0.1% parameters, Prefix-tuning outperforms other lightweight baselines and achieves a comparable performance with fine-tuning. The best score is boldfaced for both GPT-2$_{\text{MEDIUM}}$ and GPT-2$_{\text{LARGE}}$.



Prefix-Tuning: Optimizing Continuous Prompts for Generation https://aclanthology.org/2021.acl-long.353/

Low-Rank Adaptation (LoRA)

Low-Rank Adaptation Learns a low-rank "diff" between the pretrained and finetuned weight matrices. Easier to learn than prefix-tuning.



composition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, our modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{3}$$

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS: https://arxiv.org/pdf/2106.09685.pdf

Suppose a transformer layer has a **weight matrix** $W \in \mathbb{R}^{4\times4}$ that maps 4D input vectors to 4D outputs. For example:

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is an identity matrix — a very basic starting point. Let's say this is from a **pre-trained model** and we want to **fine-tune it for a specific task**.

⚠️ Full fine-tuning (baseline)

Full fine-tuning would mean learning a completely new matrix $W_{\text{new}} \in \mathbb{R}^{4\times4}$. Suppose after training we get:

$$W_{\text{new}} = \begin{bmatrix} 1.2 & 0.1 & 0.0 & -0.1 \\ -0.1 & 0.9 & 0.2 & 0.0 \\ 0.0 & 0.0 & 1.1 & 0.1 \\ 0.0 & -0.1 & 0.0 & 1.0 \end{bmatrix}$$

We had to store and train all 16 parameters.

Turning to ChatGPT for a toy example ☺

Instead of learning all 16 entries, **LoRA freezes** W and learns **a low-rank update**:

$$W_{\text{LoRA}} = W + \Delta W$$

And $\Delta W = AB$, where:

- $A \in \mathbb{R}^{4\times r}$
- $B \in \mathbb{R}^{r\times4}$
- with **rank** $r = 1$ (just one dimension!)

Let's say during training, LoRA learns:

$$A = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.0 \\ 0.2 \end{bmatrix}, \quad B = \begin{bmatrix} 0.5 & 0.1 & -0.2 & 0.0 \end{bmatrix}$$

Then:

$$\Delta W = AB = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.0 \\ 0.2 \end{bmatrix} \begin{bmatrix} 0.5 & 0.1 & -0.2 & 0.0 \end{bmatrix} = \begin{bmatrix} 0.05 & 0.01 & -0.02 & 0.0 \\ -0.05 & -0.01 & 0.02 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.10 & 0.02 & -0.04 & 0.0 \end{bmatrix}$$

Then the final adapted weight is:

$$W_{\text{LoRA}} = W + \Delta W$$

That's:

$$\begin{bmatrix} 1.05 & 0.01 & -0.02 & 0.0 \\ -0.05 & 0.99 & 0.02 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.10 & 0.02 & -0.04 & 1.0 \end{bmatrix}$$

So with only **8 parameters** (4 in A, 4 in B), we achieve an **effective fine-tuning** close to full fine-tuning — but **much cheaper**.

LoRA

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

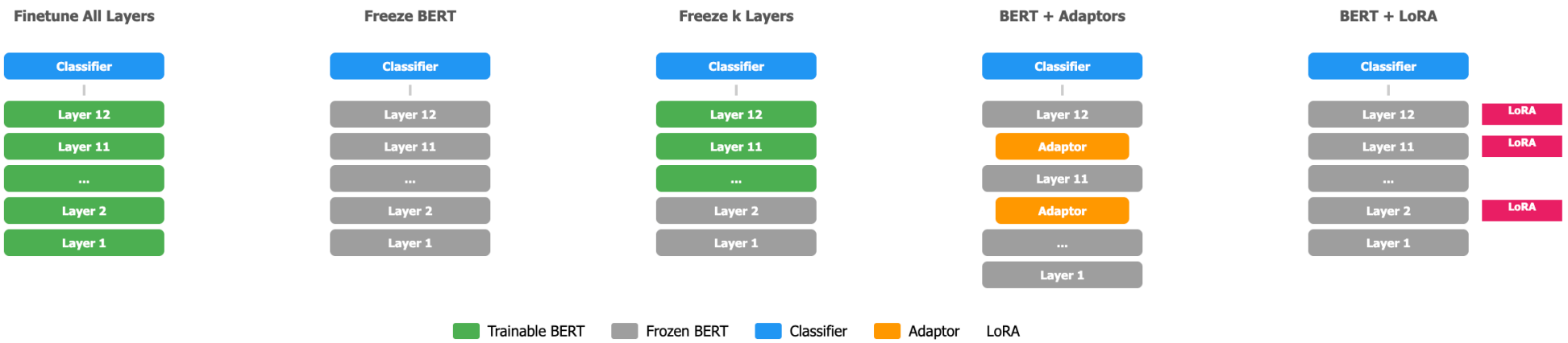
LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS: https://arxiv.org/pdf/2106.09685.pdf

Finetuning BERT for classification

We typically do one of the followings:

- Finetune all layers of BERT while training the classifier
- Freeze BERT and train only the classifier
- Freeze k layers of BERT and finetune the rest while training the classifier
- Freezing BERT and training Adaptors while training the classifier
- Inject LoRA and training the weights along with the classifier

Finetuning BERT for classification (Hands-on)



Here is a notebook. Try these techniques in Google Colab.