Generative Pre-trained Transformer (GPT)

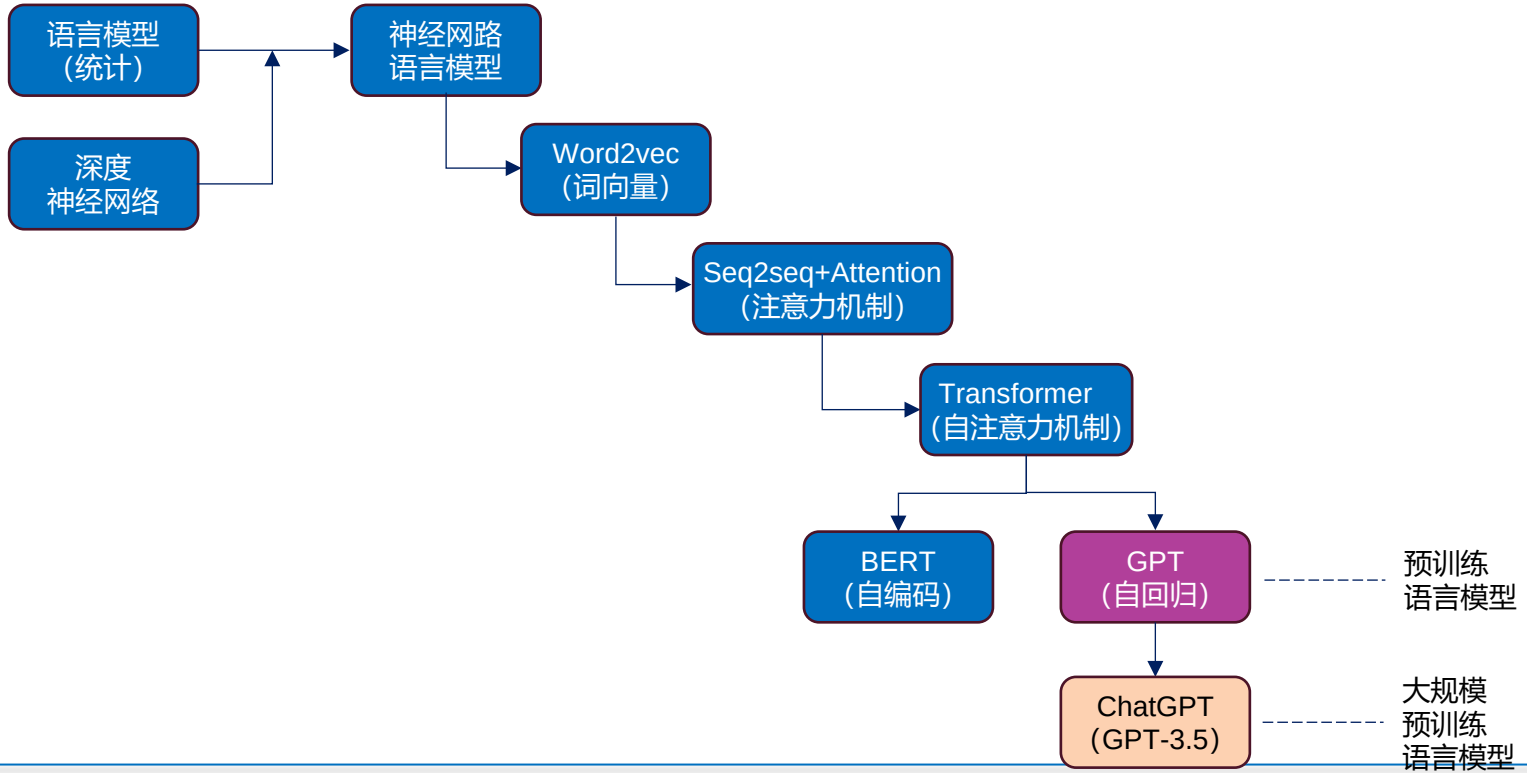
Haihua Xie, Assistant Professor

Beijing Institute of Mathematical Sciences and Applications (BIMSA)

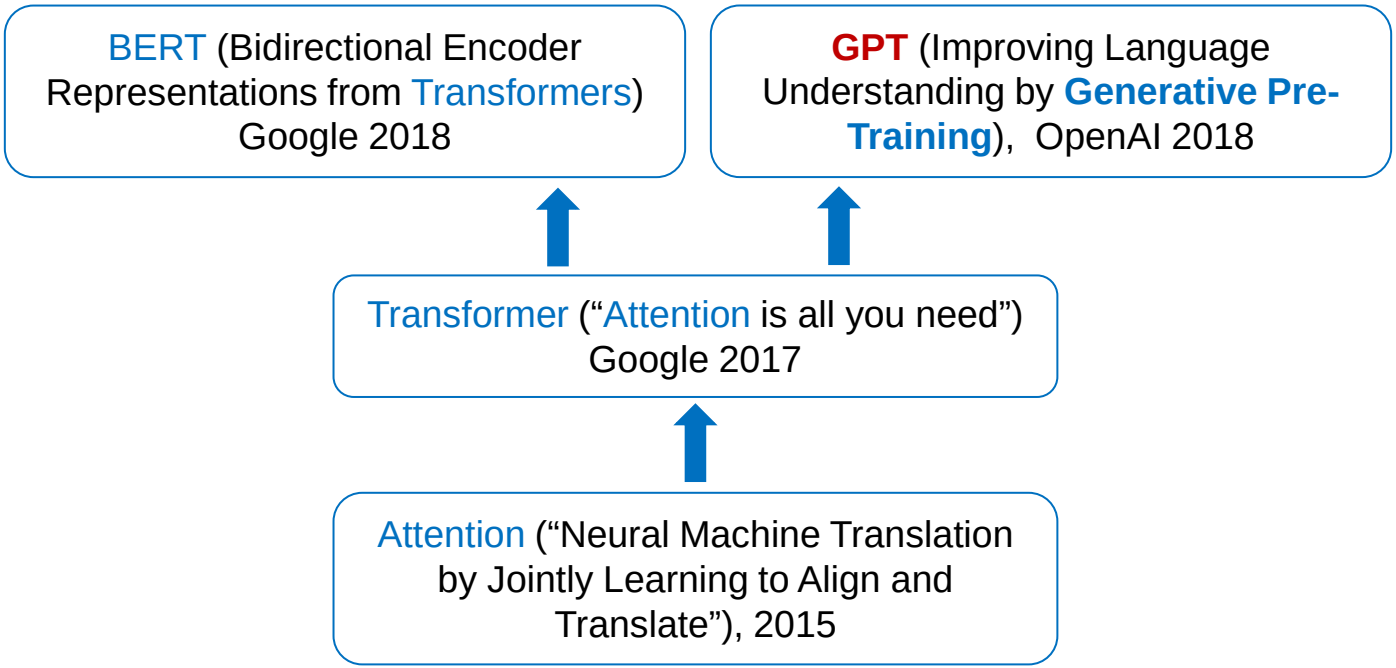
Artificial Intelligence and Machine Learning

2024.05

语言模型发展脉络


语言模型
(统计)
深度
神经网络
神经网路
语言模型
Word2vec
(词向量)
Seq2seq+Attention
(注意力机制)
Transformer
(自注意力机制)
BERT
(自编码)
GPT
(自回归)
预训练
语言模型
ChatGPT
(GPT-3.5)
大规模
预训练
语言模型

GPT: Generative Pre-Training



Brief Introduction to GPT-2

GPT-2

- A very large, transformer-based language model trained on a massive dataset.
- Architecture very similar to the decoder-only transformer.

OpenAI testers gave the new engine an absurdist opener:

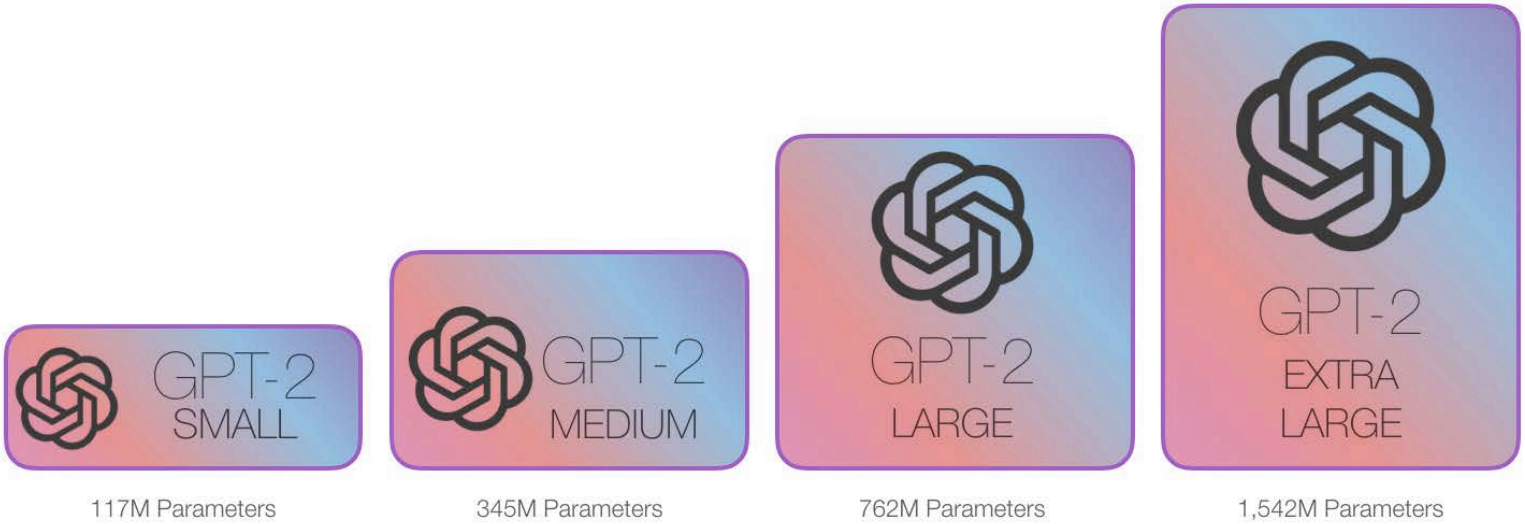
"In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English."

The engine returned with this: "The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Perez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans.

Brief Introduction to GPT-2

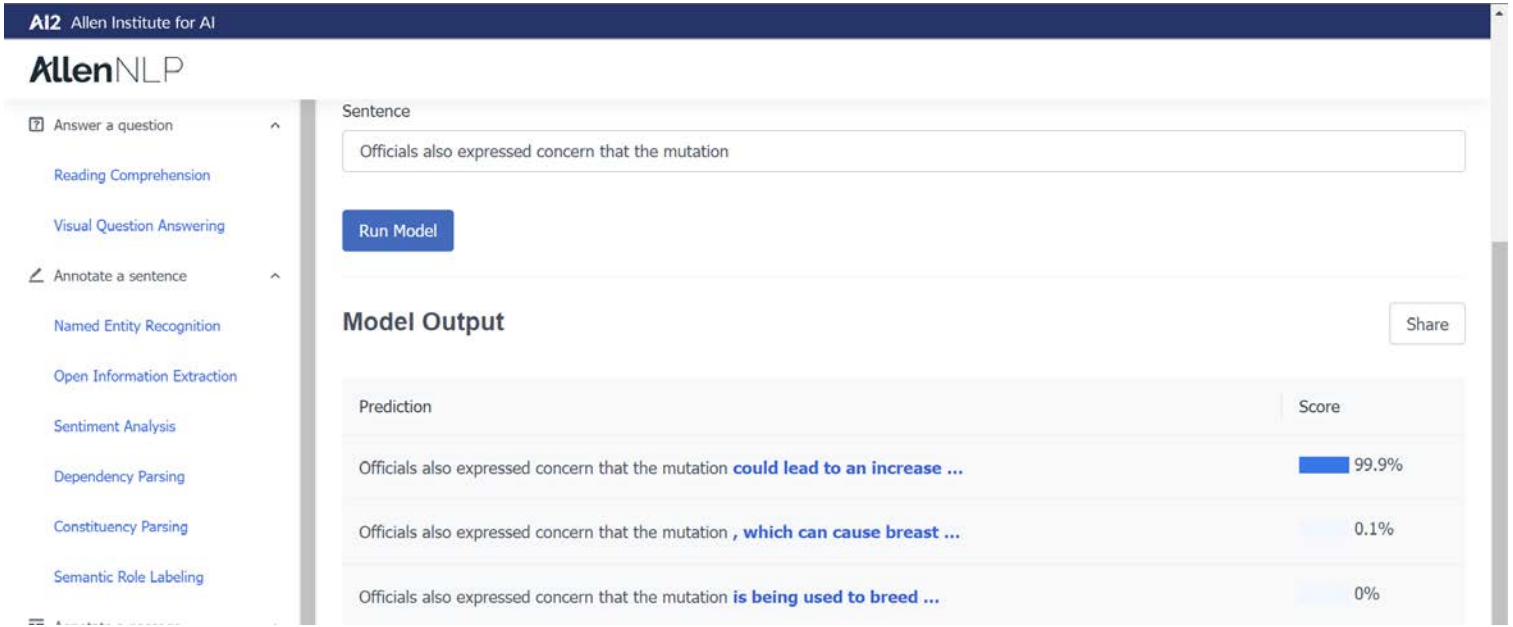
GPT-2

- Trained on a massive 40GB dataset called WebText.
- Takes up 500MBs ~ 6.5 GBs of storage to store all of its parameters.



Brief Introduction to GPT-2

https://demo.allennlp.org/next-token-lm

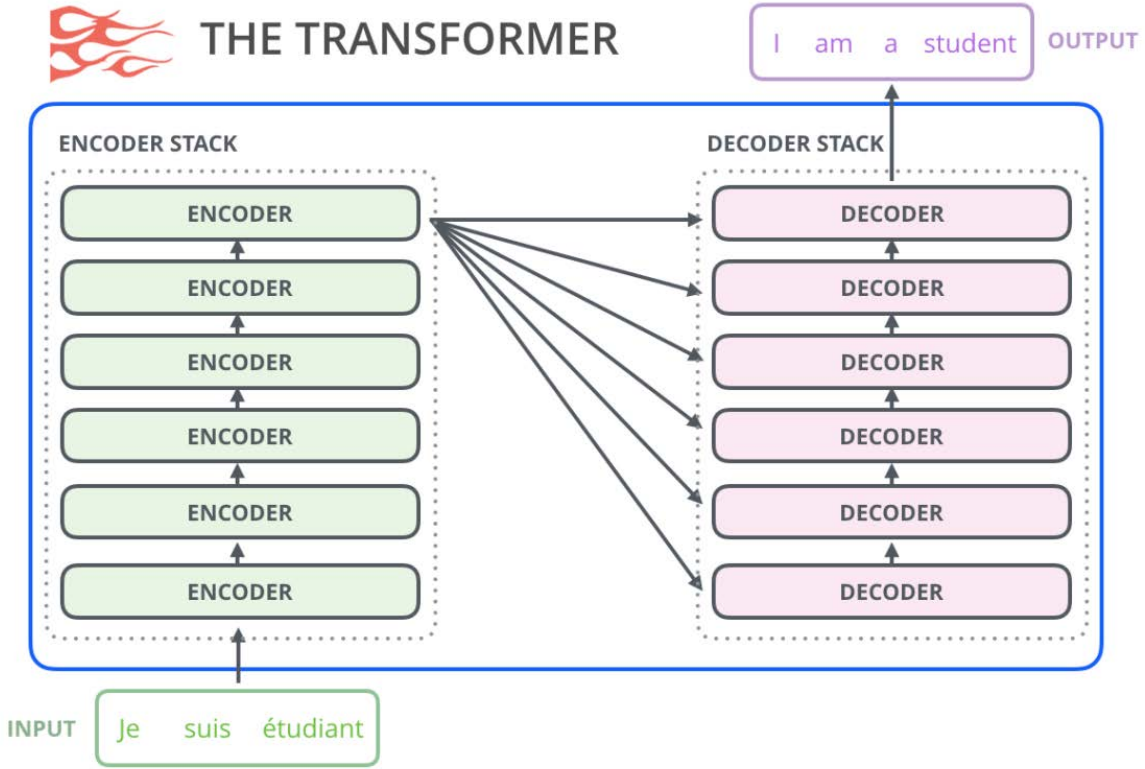


Officials also expressed concern that the mutation could result in immune evasion and enhanced transmissibility of the virus.

Transformers for Language Modeling

Original transformer

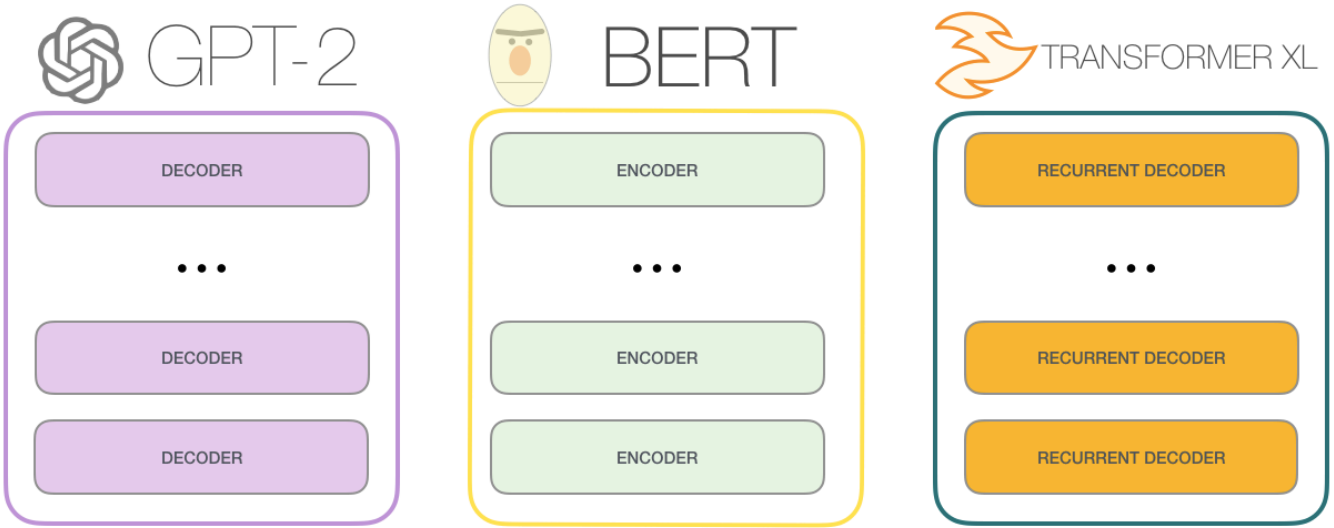
- Encoder-decoder architecture.
- Suitable for text generation (e.g., machine translation) tasks.



Transformers for Language Modeling

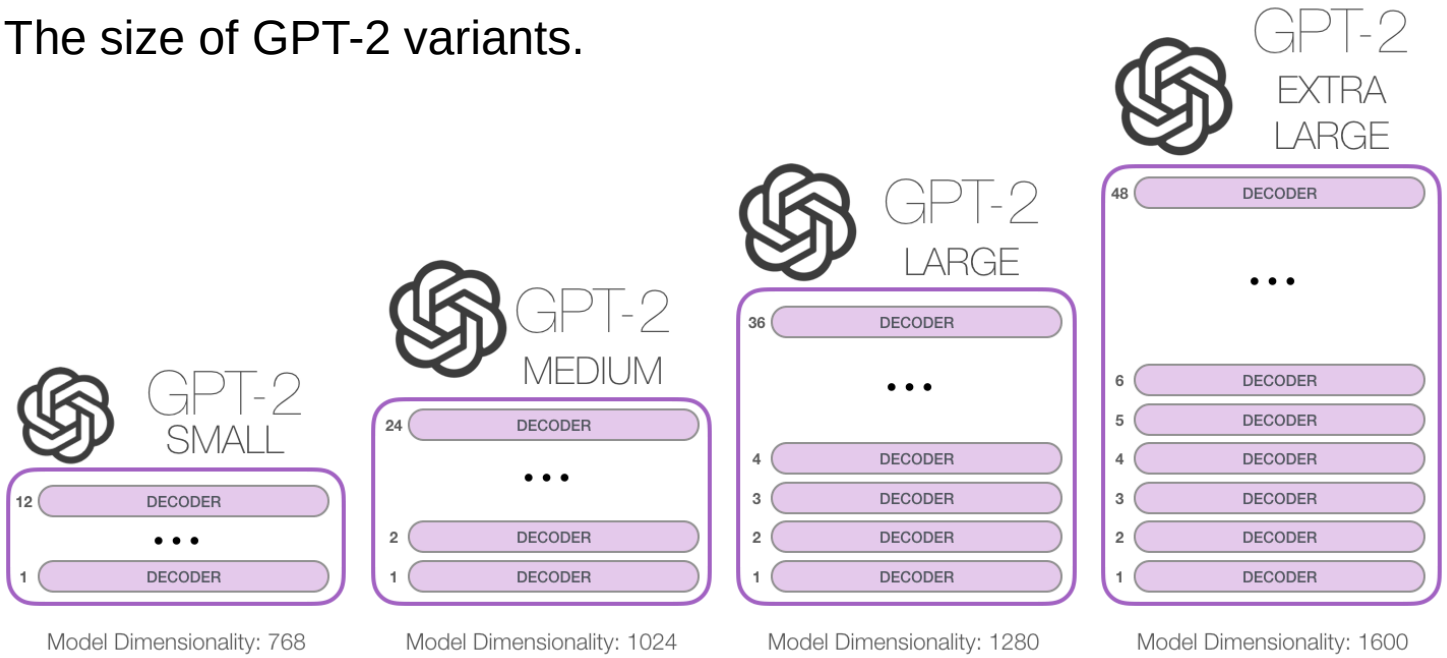
Large Language Models (LLM)

- The architecture shed either the encoder or decoder, stacking as high as possible
- Feed with massive amounts of training text
- Throw vast amounts of compute at them
- Cost hundreds of thousands of dollars to train



Transformers for Language Modeling

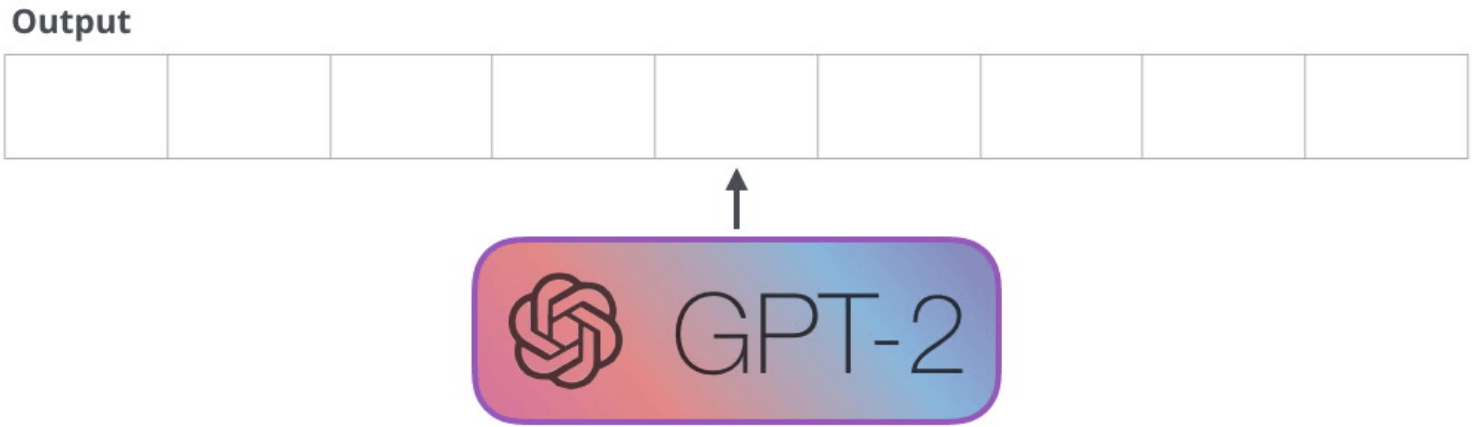
The size of GPT-2 variants.



One Difference From BERT

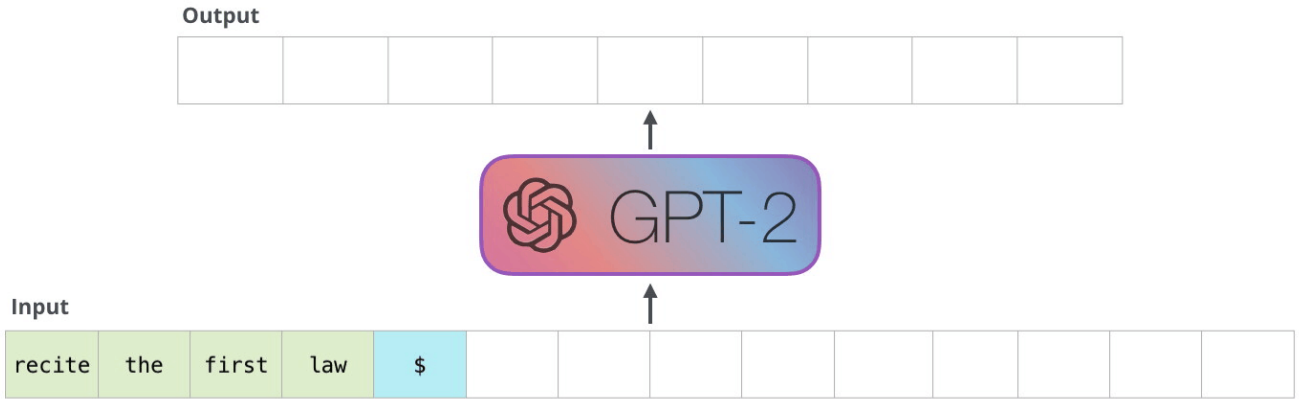
The GPT-2 is built using transformer decoder blocks. BERT, on the other hand, uses transformer encoder blocks.

GPT2, like traditional language models, outputs one token at a time.



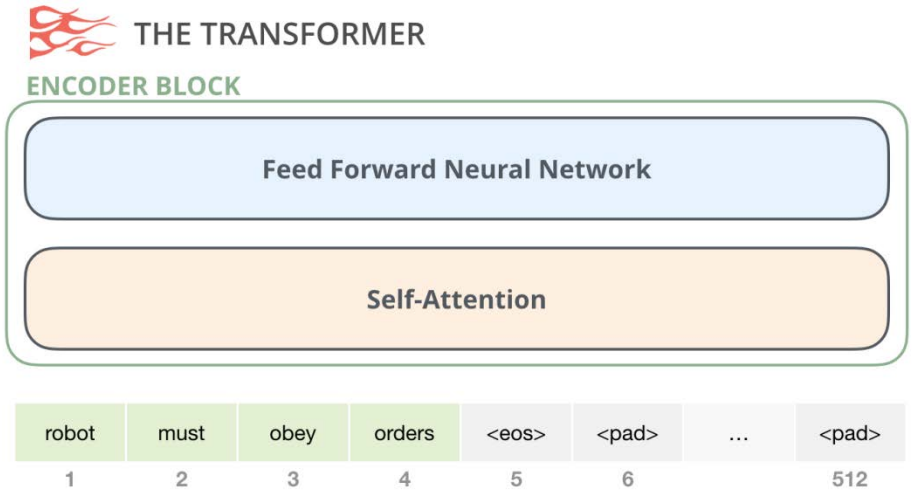
One Difference From BERT

Auto-regression: After each token is produced, that token is added to the sequence of inputs.



The Evolution of the Transformer Block

Transformer Encoder Block

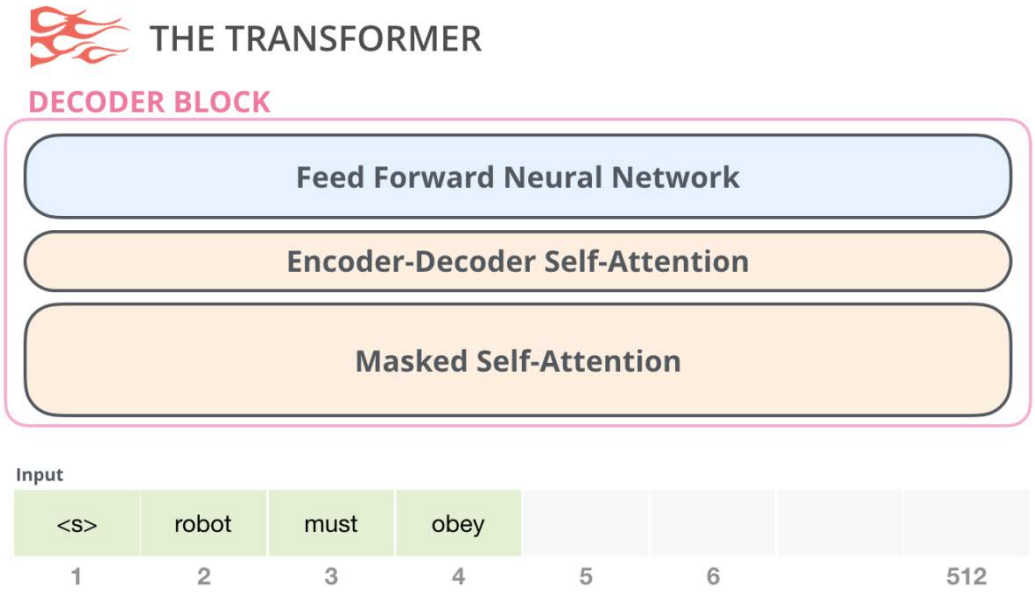


An encoder block can take inputs up until a certain max sequence length (e.g. 512 tokens).

The Evolution of the Transformer Block

Transformer Decoder Block

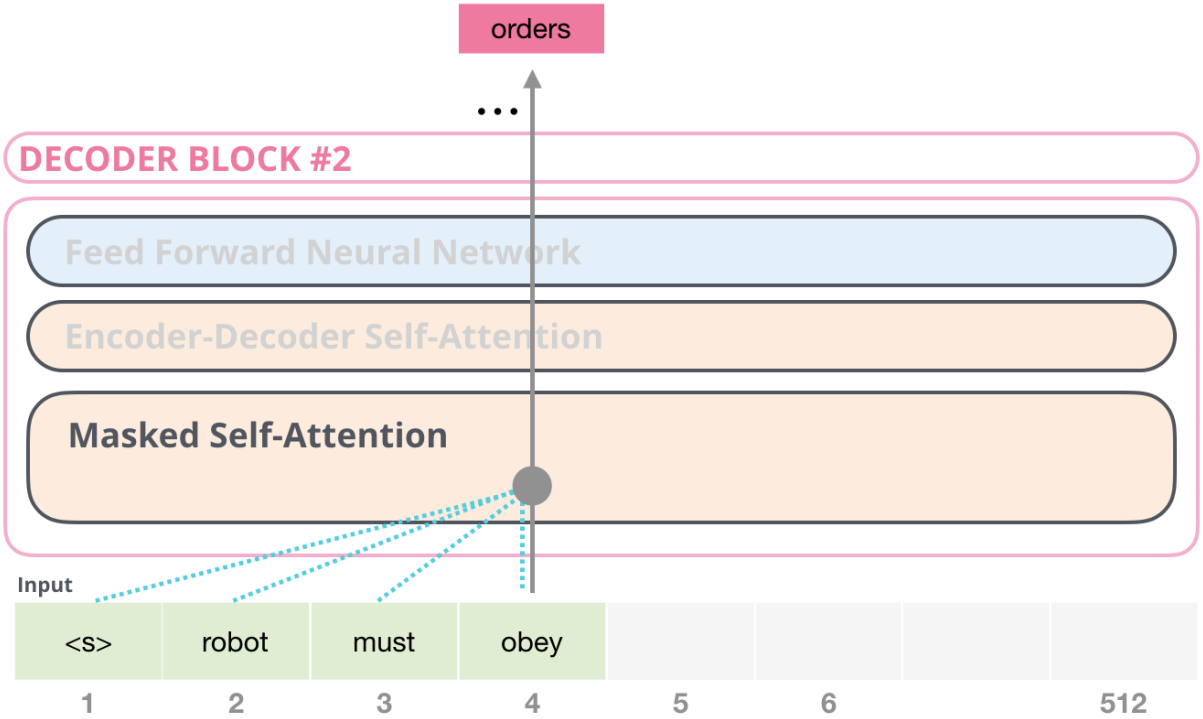
The decoder block has a layer to allow it to pay attention to specific segments from the encoder:



The Evolution of the Transformer Block

The self-attention calculation blocks information from tokens that are to the right of the position being calculated.

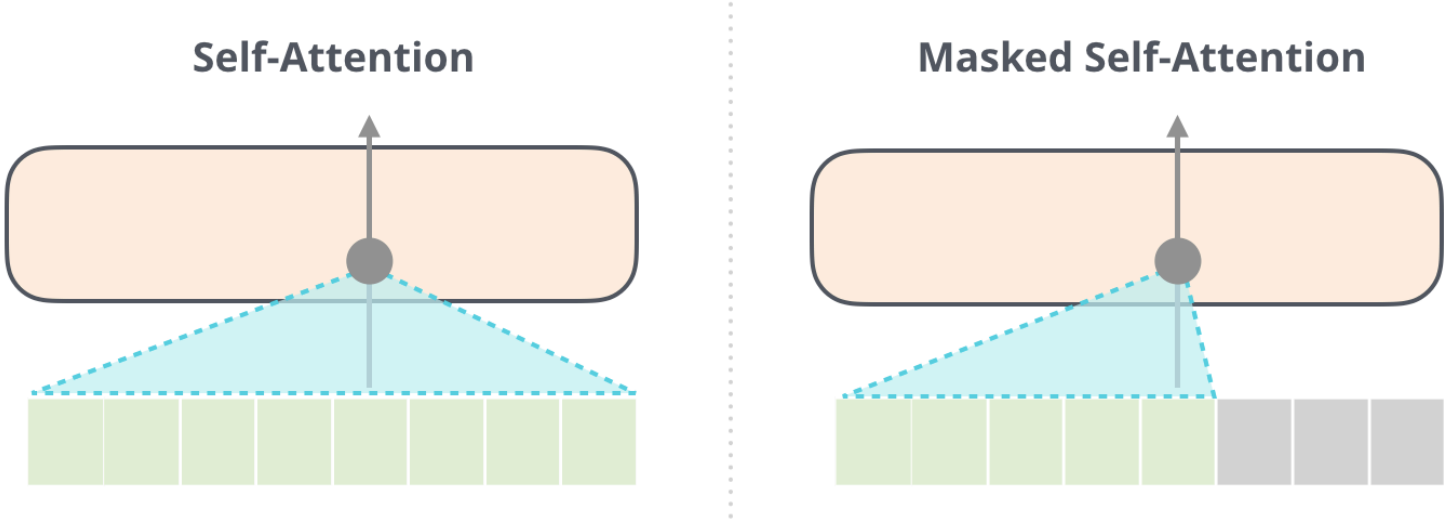
To highlight the path of position #4, we can see that it is only allowed to attend to the present and previous tokens:



The Evolution of the Transformer Block

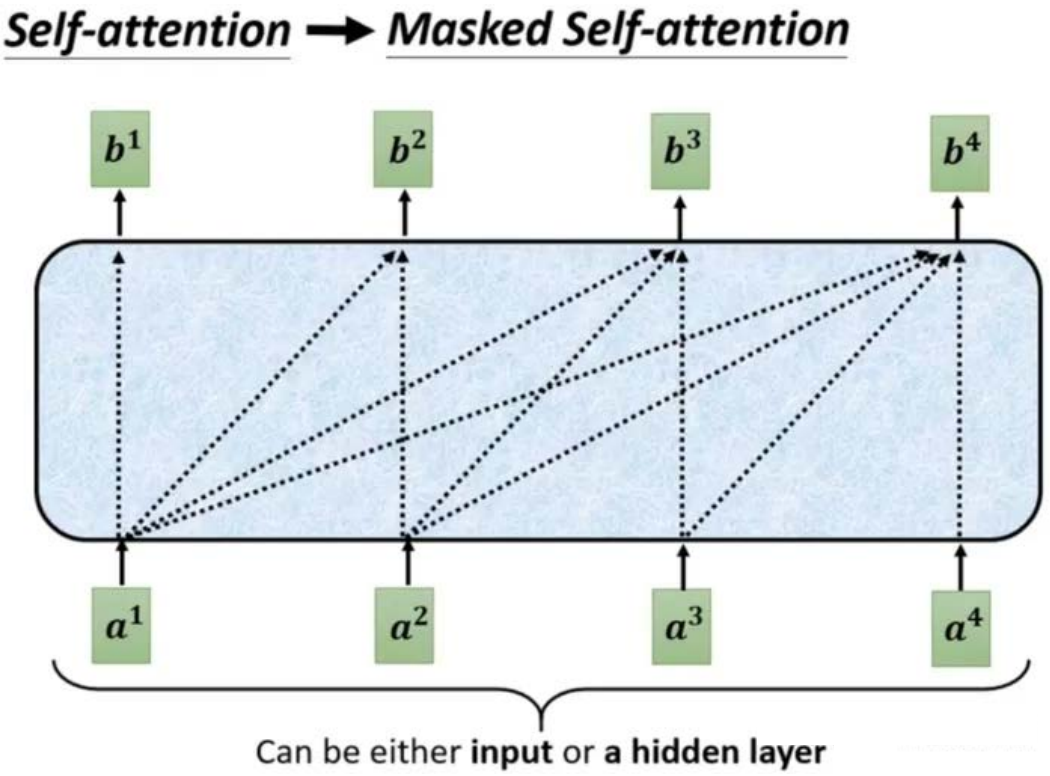
A normal self-attention block allows a position to peak at tokens to its right.

Masked self-attention prevents that from happening:



The Evolution of the Transformer Block

Masked self-attention



The Evolution of the Transformer Block

Masked Self-attention

第二步，计算<sos>的（每一层）向量值时，不考虑词1（即没有Attention(0,1)的计算）：

- <sos>的（每一层）向量值直接采用前一步的计算结果，以减小计算量。

	输入	Attention	预测
第一步	<sos>	Attention(0,0)	<词1>
第二步	<sos> <词1>	Attention(0,0) Attention(1,0) Attention(1,1)	<词2>

在训练阶段，语句（<sos>，词1，词2，词3）一次性输入模型，并行预测<词1>，<词2>，<词3>。此时采用mask attention以确保预测<词x>时不会有已知的信息。

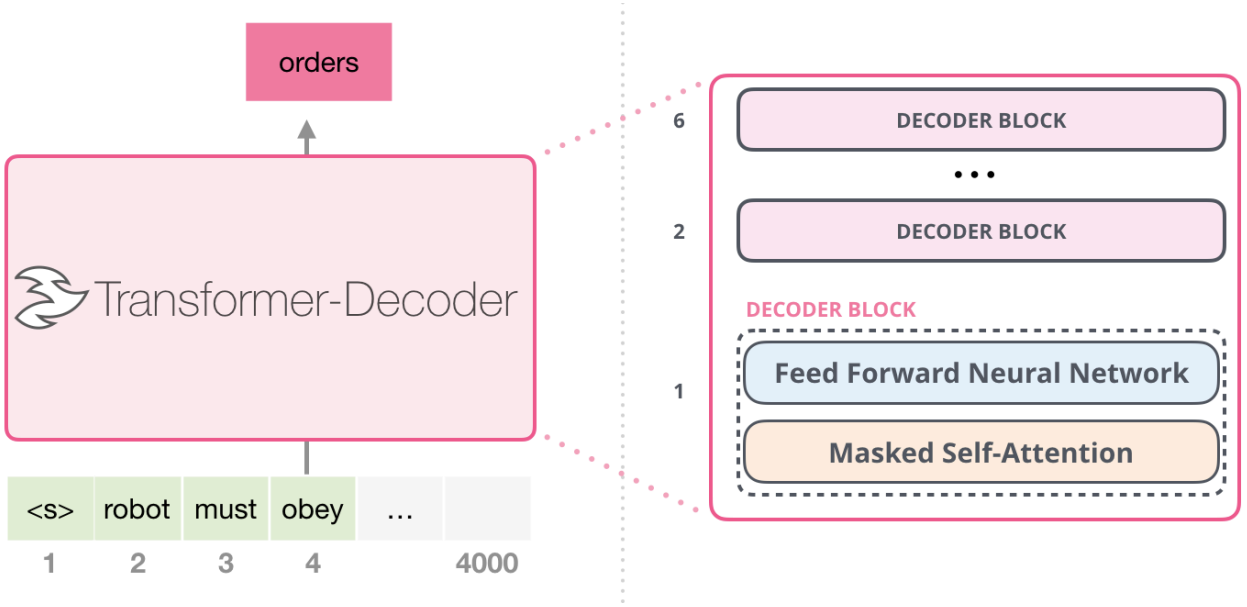
	<sos>	<词1>	<词2>	<词3>	预测
<sos>	*	o	o	o	<词1>
<词1>	*	*	o	o	<词2>
<词2>	*	*	*	o	<词3>
<词3>	*	*	*	*	<eos>

The Evolution of the Transformer Block

The Decoder-Only Block

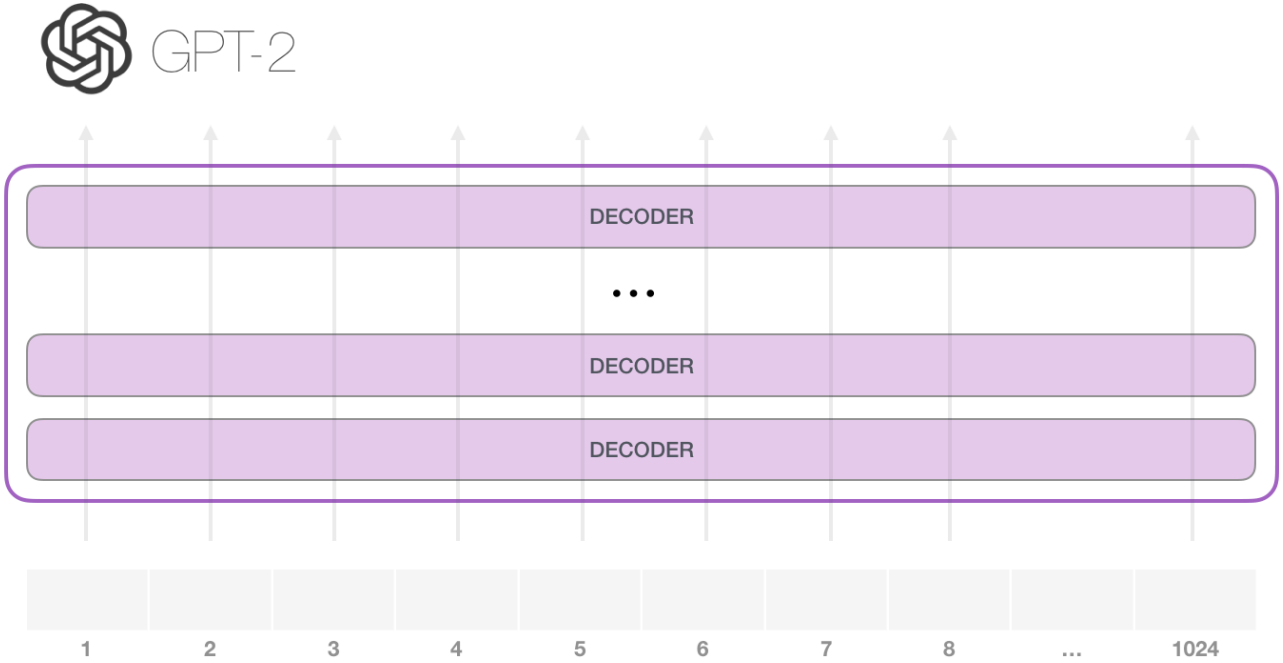
This early transformer-based language model was made up of a stack of six transformer decoder blocks:

These blocks were very similar to the original decoder blocks, except they did away with that decoder-encoder attention layer.



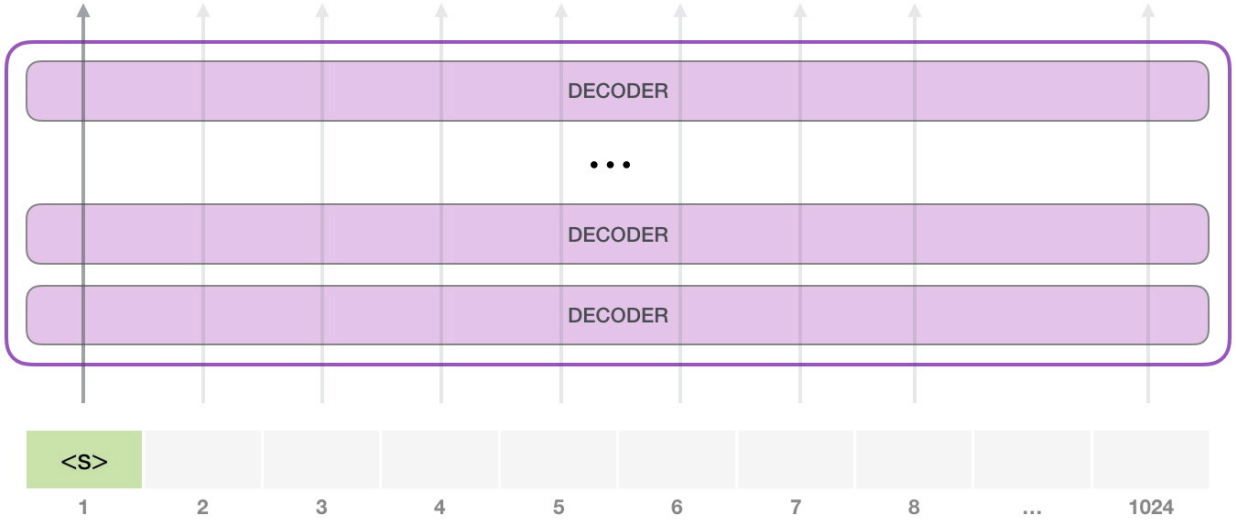
Looking Inside GPT-2

In GPT-2, each token flows through all the decoder blocks along its own path.



Looking Inside GPT-2

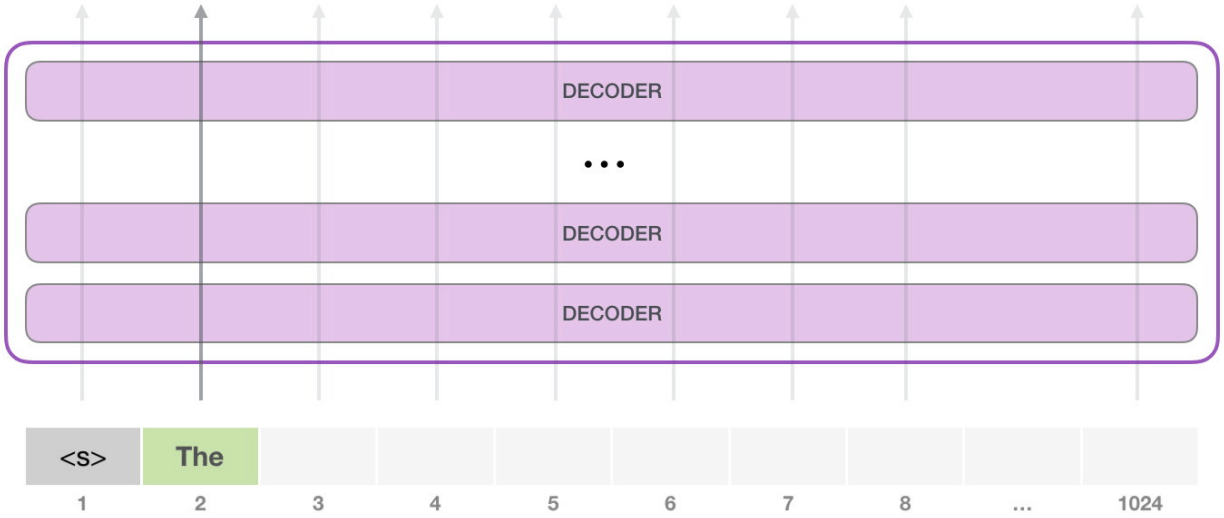
1) The vector of **<s>** is scored against the model's vocabulary (50,000 words in the case of GPT-2).



Looking Inside GPT-2

2) Add the output from the first step to the input sequence, and have the model make its next prediction.

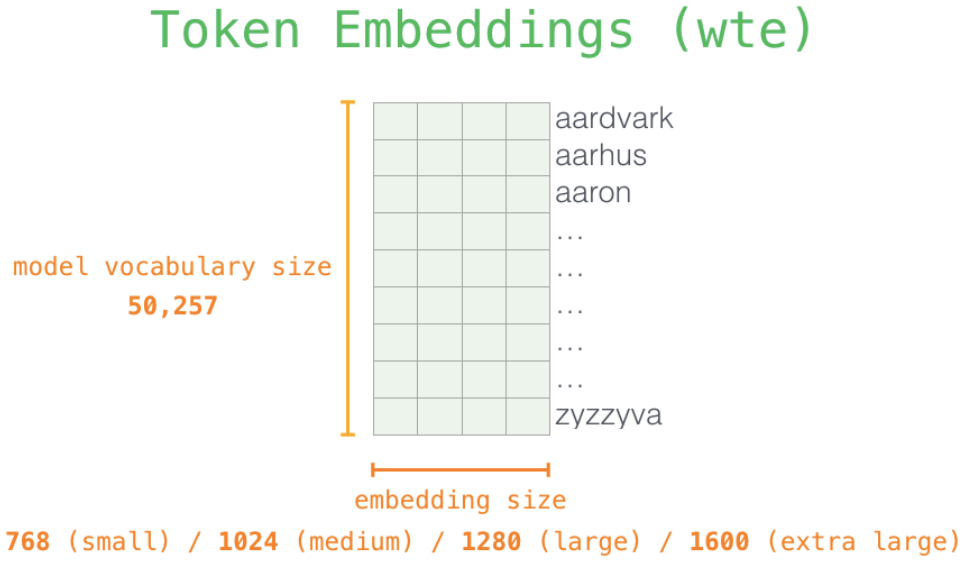
Each layer of GPT-2 has retained its own interpretation of the first token and will use it in processing the second token. GPT-2 does not re-interpret the first token in light of the second token.



A Deeper Look Inside

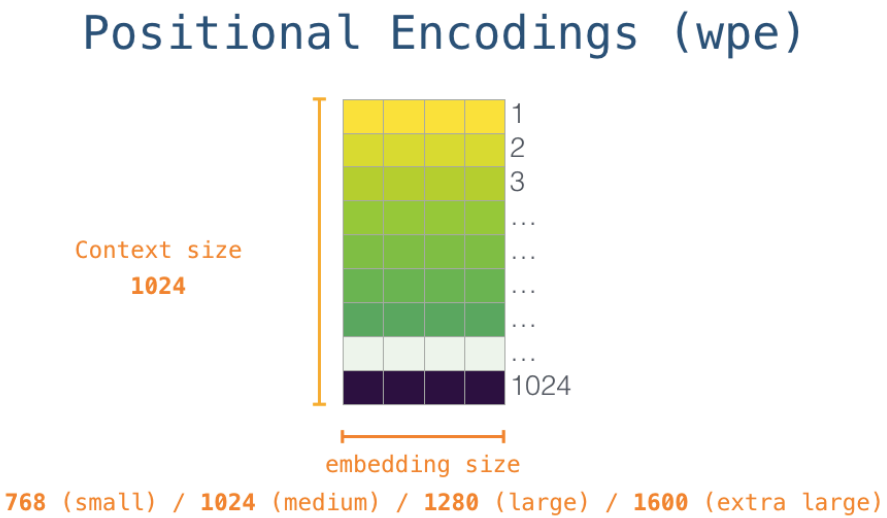
Input Encoding

The model looks up the embedding of the input word in its embedding matrix – one of the components we get as part of a trained model.



A Deeper Look Inside

Positional encoding: Part of the trained model, a matrix that contains a positional encoding vector for each of the 1024 positions in the input.



A Deeper Look Inside

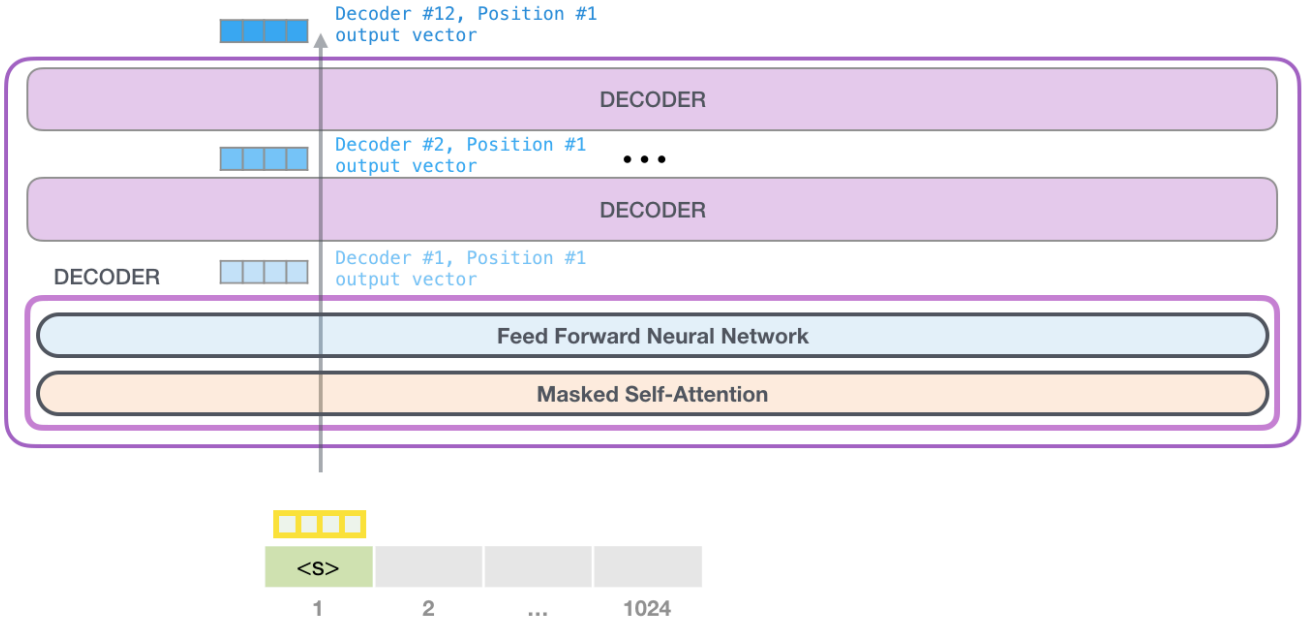
Token embedding and positional encoding: Two of the weight matrices that constitute the trained GPT-2.

Sending a word to the first transformer block means looking up its embedding and adding up the positional encoding vector for position #1.

A Deeper Look Inside

A journey up the Stack

- Block by block, self-attention → FFNN.
- Each block has the same structure but own weights.



A Deeper Look Inside

Self-Attention Recap

Language heavily relies on context. For example, look at the second law:

> ***Second Law of Robotics***
> *A robot must obey the orders given **it** by human beings except where **such orders** would conflict with the **First Law**.*

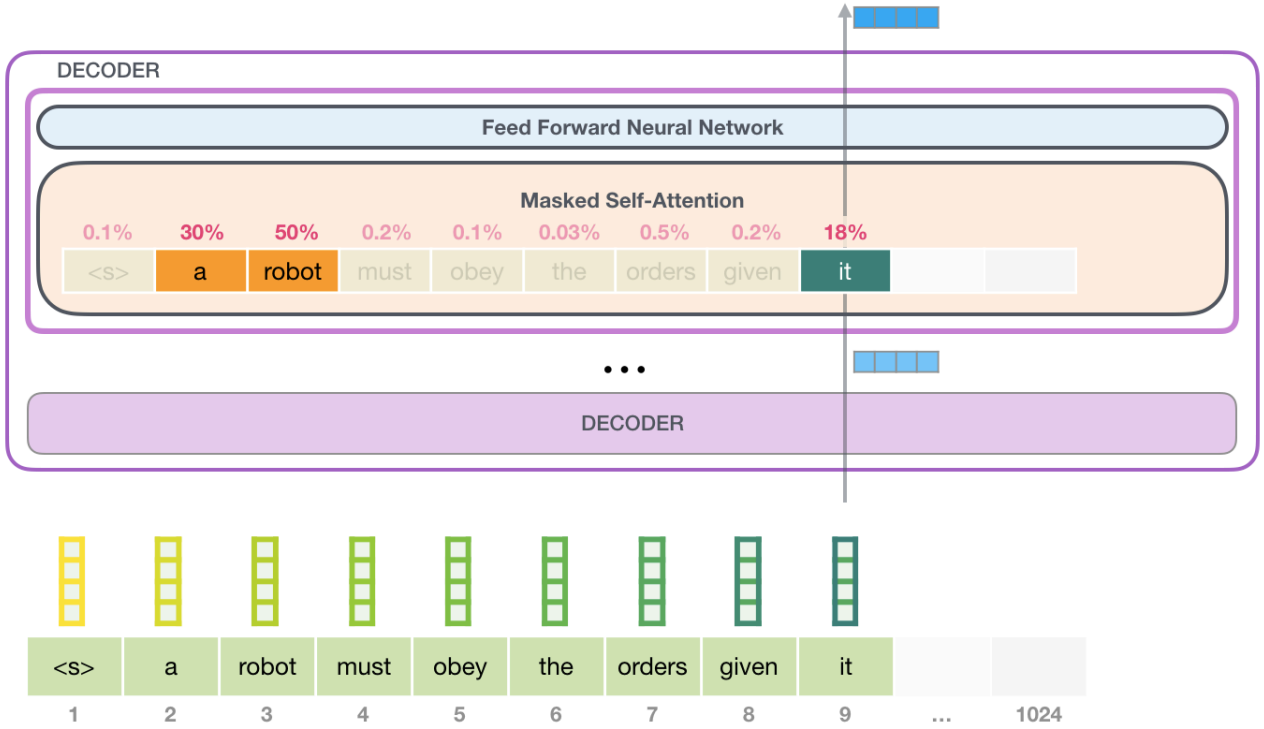
When a model processes this sentence:

- **it** refers to the robot
- **such orders** refers to the earlier part of the law, namely “the orders given it by human beings”
- **The First Law** refers to the entire First Law

Self-attention assigns scores to how relevant each word in the segment is, and adding up their vector representation.

A Deeper Look Inside

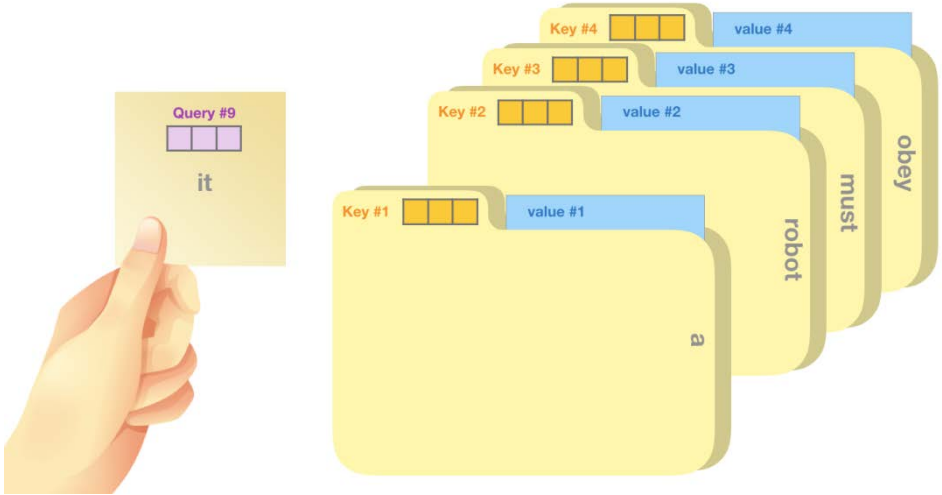
Self-attention layer in the top block is paying attention to “a robot” when it processes the word “it”.



A Deeper Look Inside

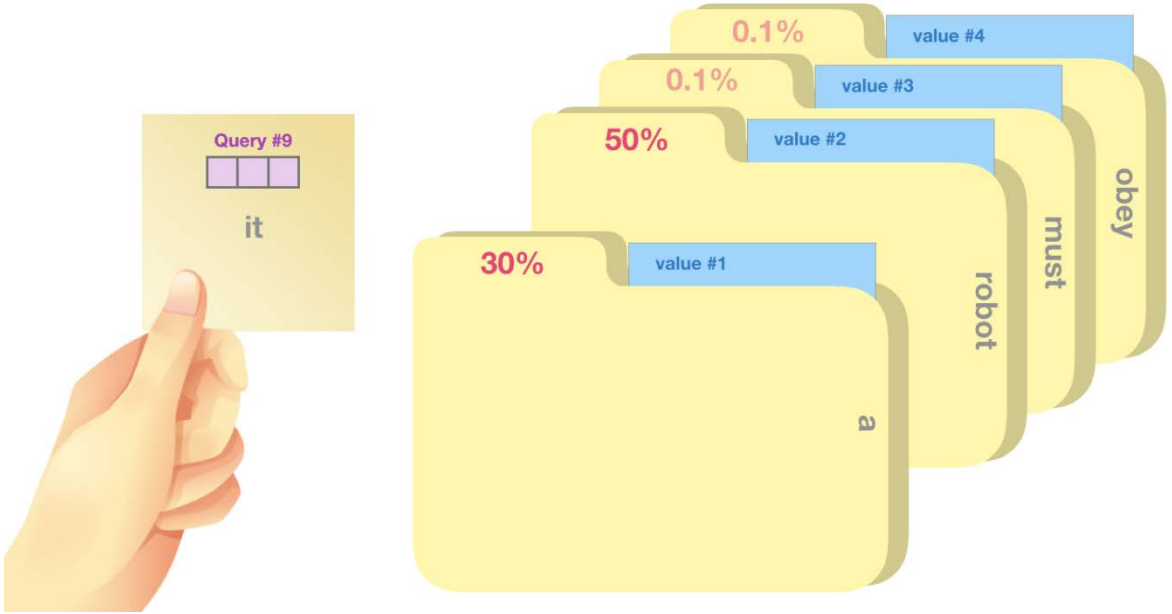
The significant components of **Self-Attention** are three vectors:

- Query: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
- Key: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
- Value: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.



A Deeper Look Inside

The query is like a sticky note with the topic you're researching.
The keys are like the labels of the folders.
The values are contents of that folder.
Multiplying the query vector by each key vector produces a score for each folder (technically: dot product followed by softmax).



A Deeper Look Inside

Self-attention outcome: multiply each value by its score and sum up.

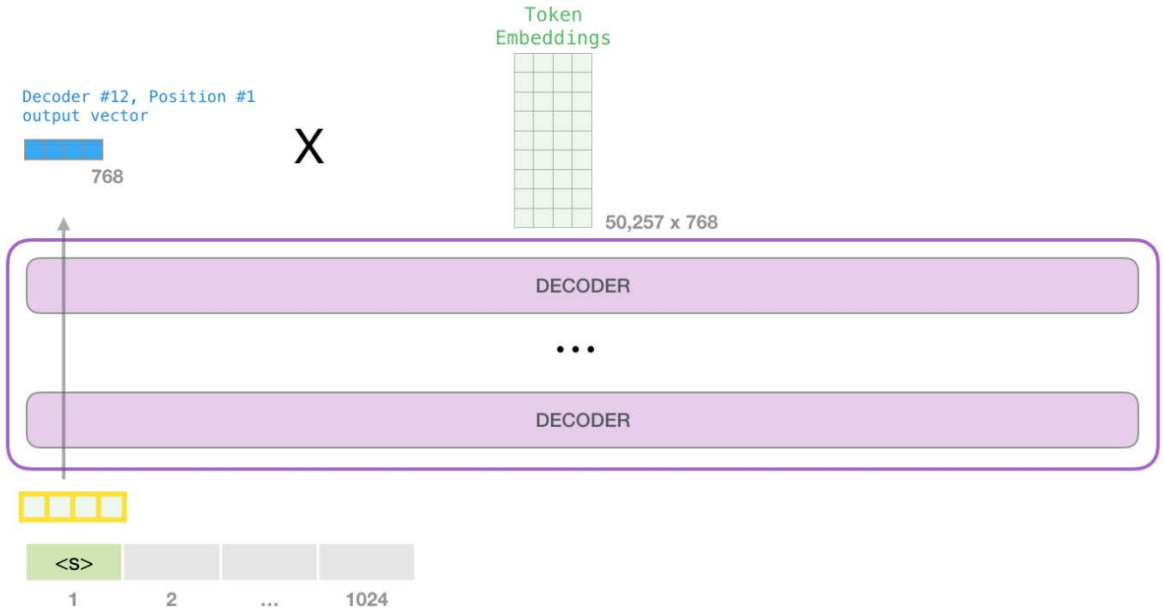
Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

This weighted blend of value vectors results in a vector that paid 50% of its “attention” to the word robot, 30% to the word a, and 19% to the word it.

A Deeper Look Inside

Model Output

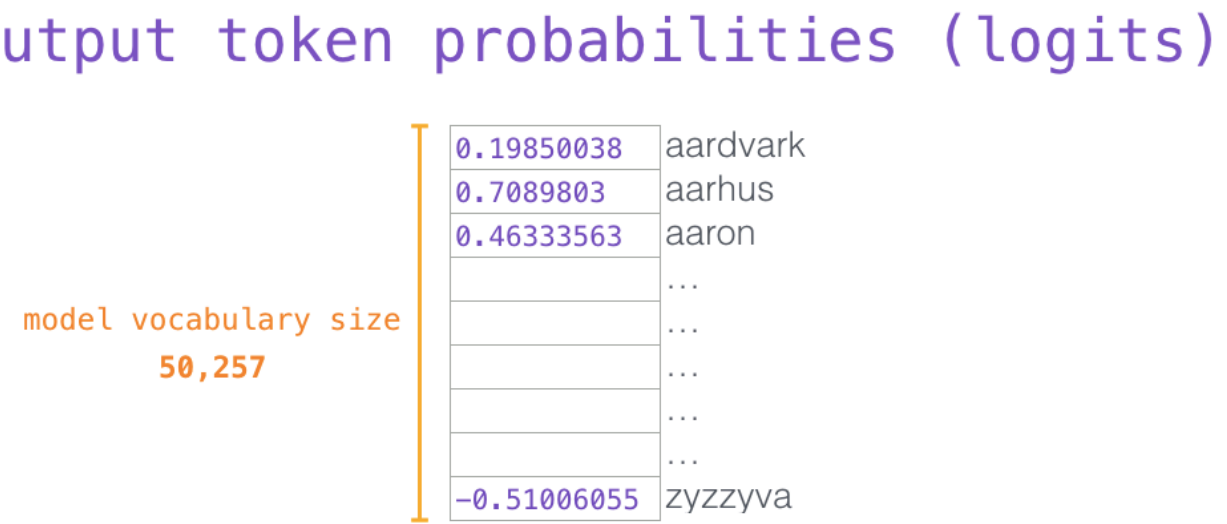
The output vector of the top block • the embedding matrix.



A Deeper Look Inside

The **embedding matrix**: the embedding of a word in the model's vocabulary.

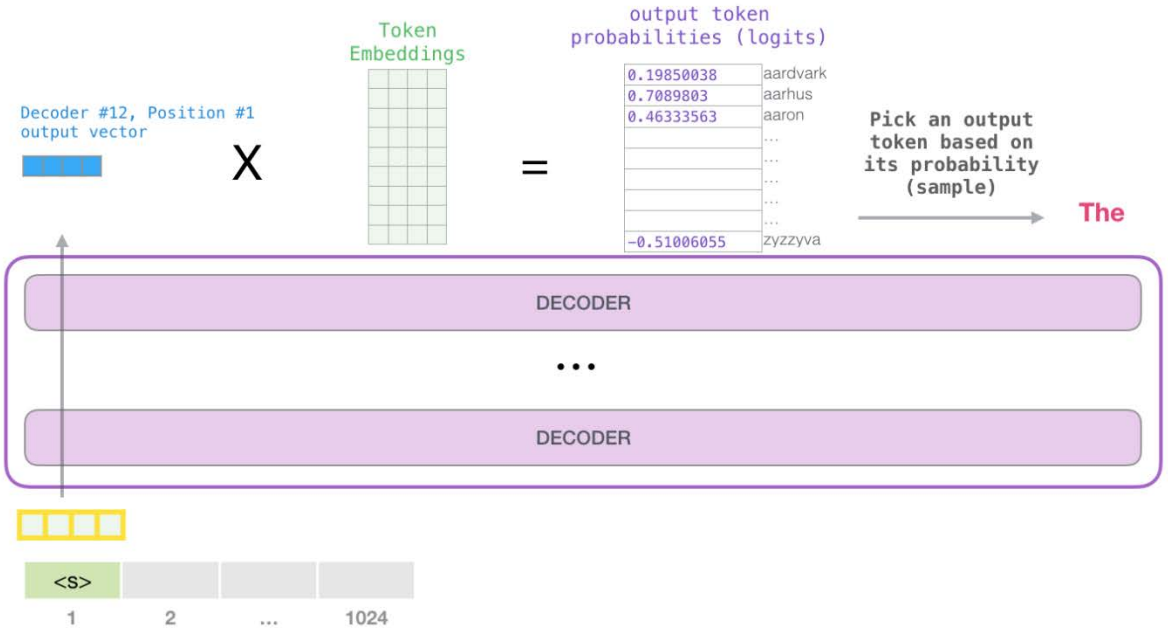
The result of this multiplication is interpreted as a score for each word.



A Deeper Look Inside

Output: the token with the highest score (**top_k = 1**).

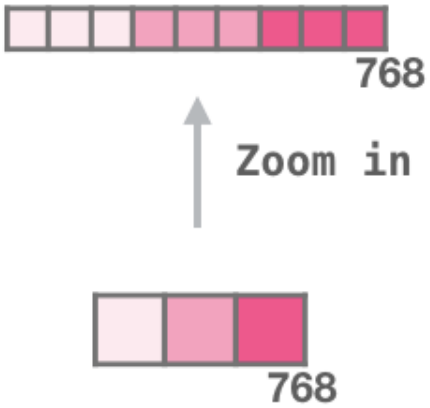
Alternative: sample a word from a list (**top_k = 40**) using the score as the probability of selecting that word (so words with a higher score have a higher chance of being selected).



The model continues iterating until the entire context is generated (1024 tokens) or until an end-of-sequence token is produced.

End of part #1: The GPT-2

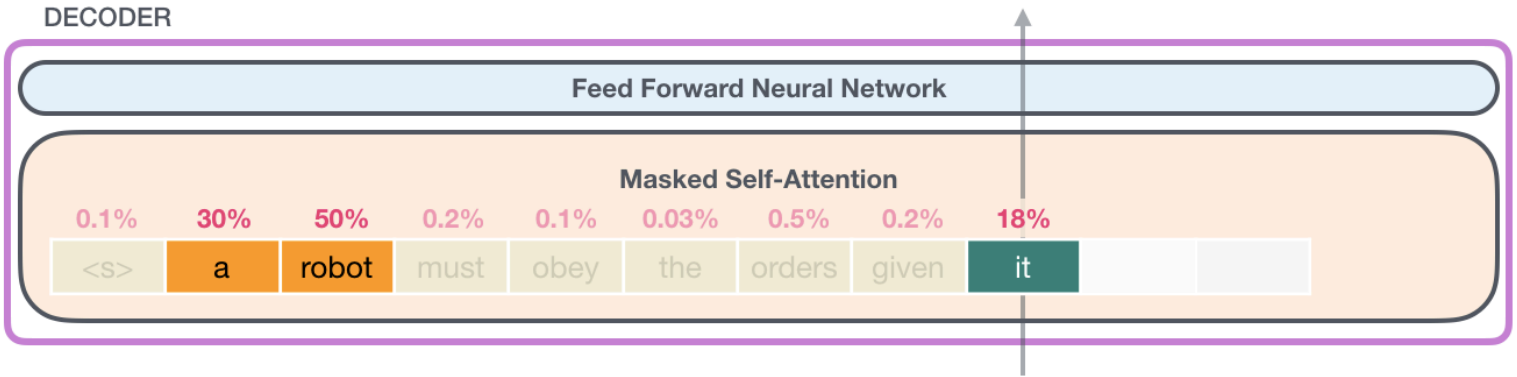
- GPT2 uses **Byte Pair Encoding** to create the tokens in its vocabulary. This means the tokens are usually parts of words.
- At training time, the model would be trained against longer sequences of text and processing multiple tokens at once. Also at training time, the model would process larger batch sizes (512) vs. the batch size of one that evaluation uses.
- Transformers use a lot of **layer normalization**.



Part #2: The Illustrated Self-Attention

Self-attention being applied in a layer that is processing the word ‘it’:

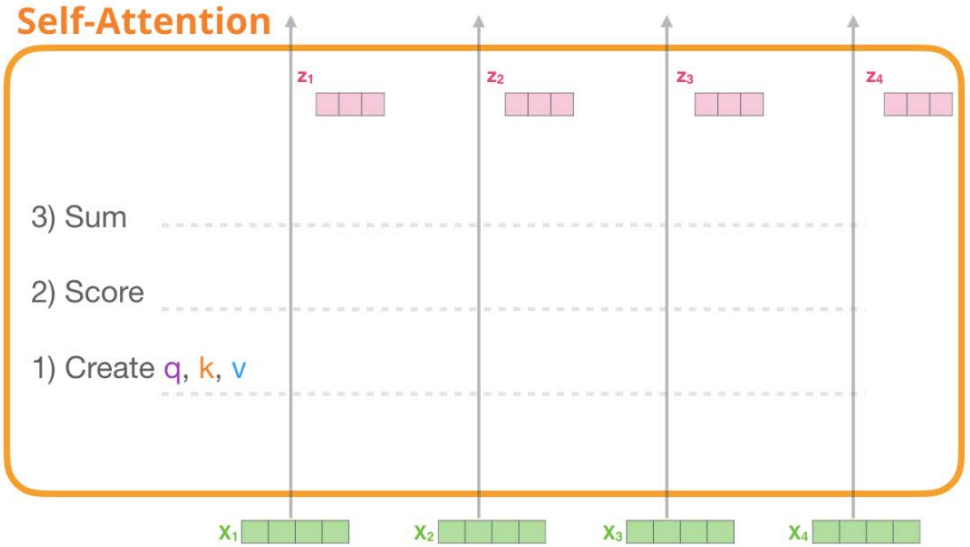
The actual implementations are done by **multiplying giant matrices together**.



Part #2: The Illustrated Self-Attention

Self-Attention (without masking)

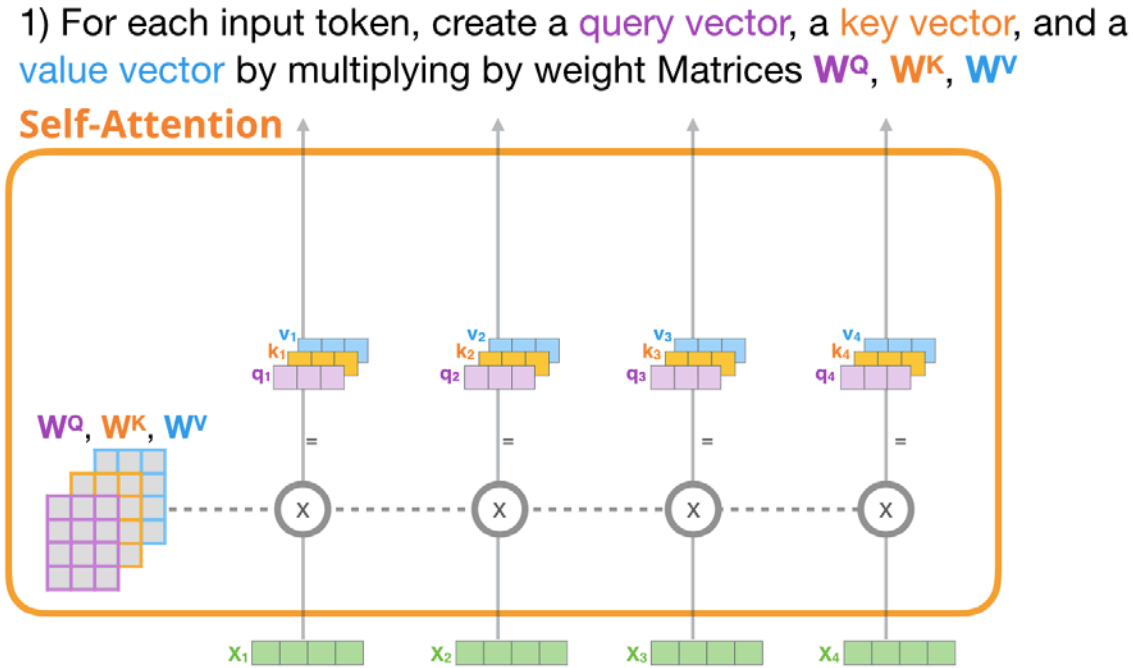
1. Create the Query, Key, and Value vectors for each path.
2. For each input token, use its query vector to score against all the other key vectors
3. Sum up the value vectors after multiplying them by their associated scores.



Part #2: The Illustrated Self-Attention

1- Create Query, Key, and Value Vectors

The first path: take its query, and compare against all the keys. That produces a score for each key. The first step in self-attention is to calculate the three vectors for each token path:

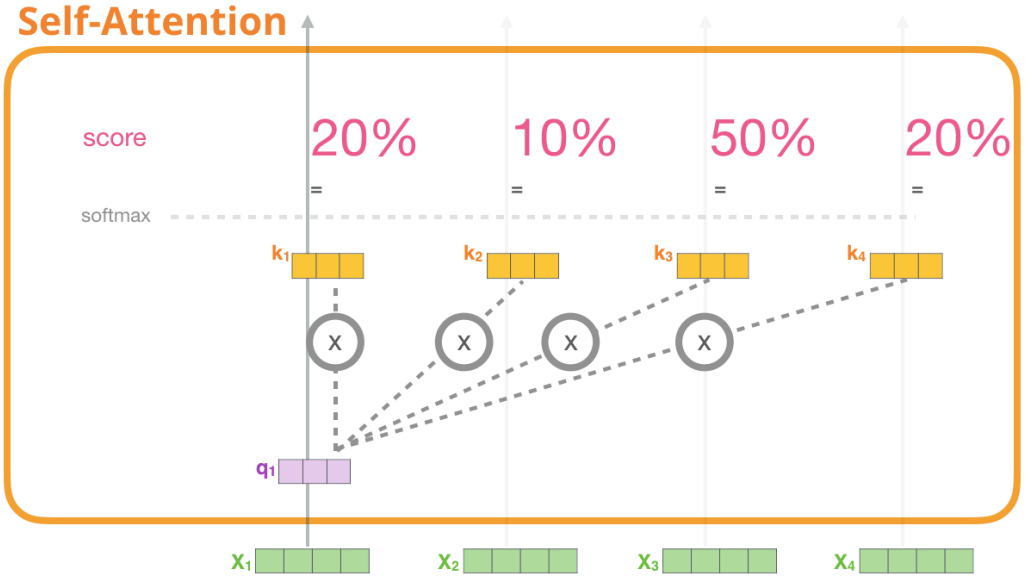


Part #2: The Illustrated Self-Attention

2- Score

Use the query and key vectors only for step #2. Multiply the query by all the other key vectors resulting in a score for each of the four tokens.

2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match

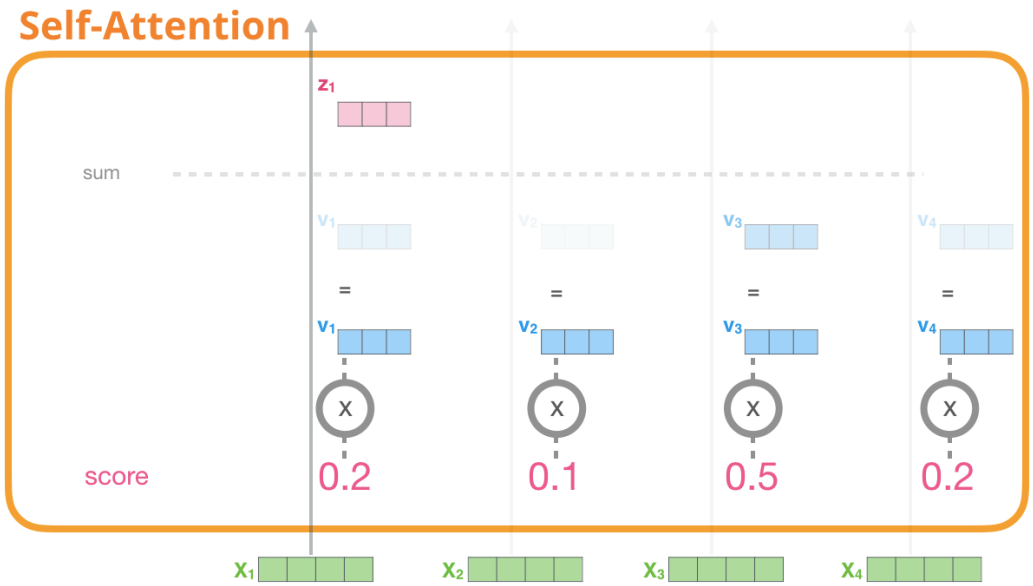


Part #2: The Illustrated Self-Attention

3- Sum

Multiply the scores by the value vectors. A value with a high score will constitute a large portion of the resulting vector after summing them up.

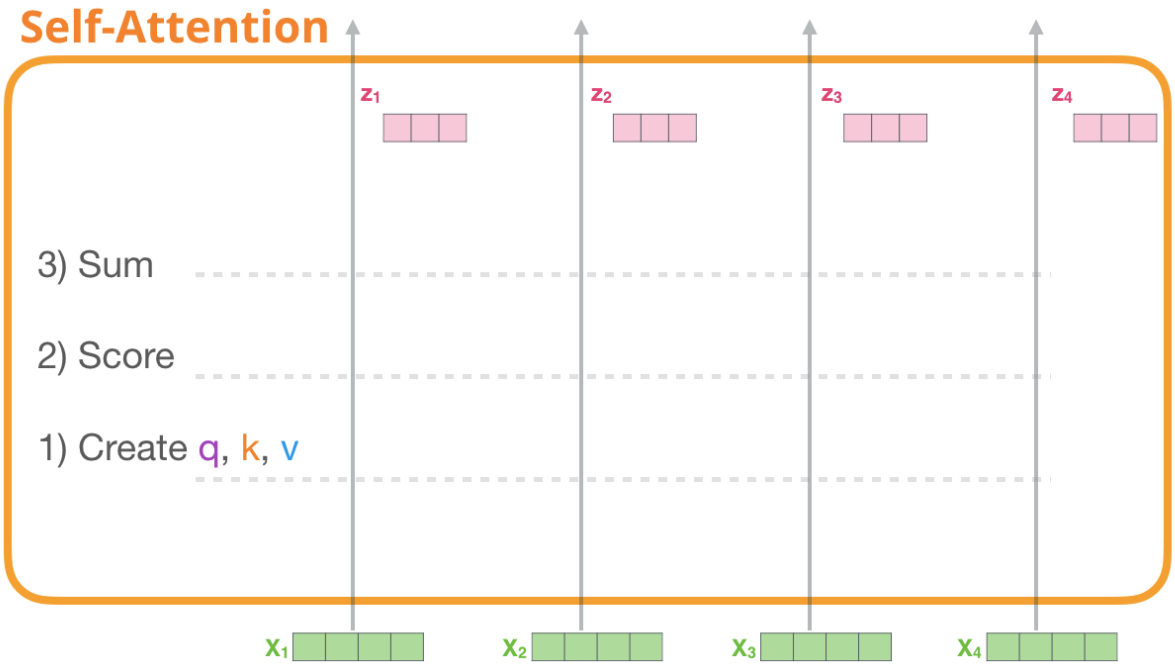
3) Multiply the value vectors by the scores, then sum up



Part #2: The Illustrated Self-Attention

End up with **a vector** representing each token containing the appropriate context of that token.

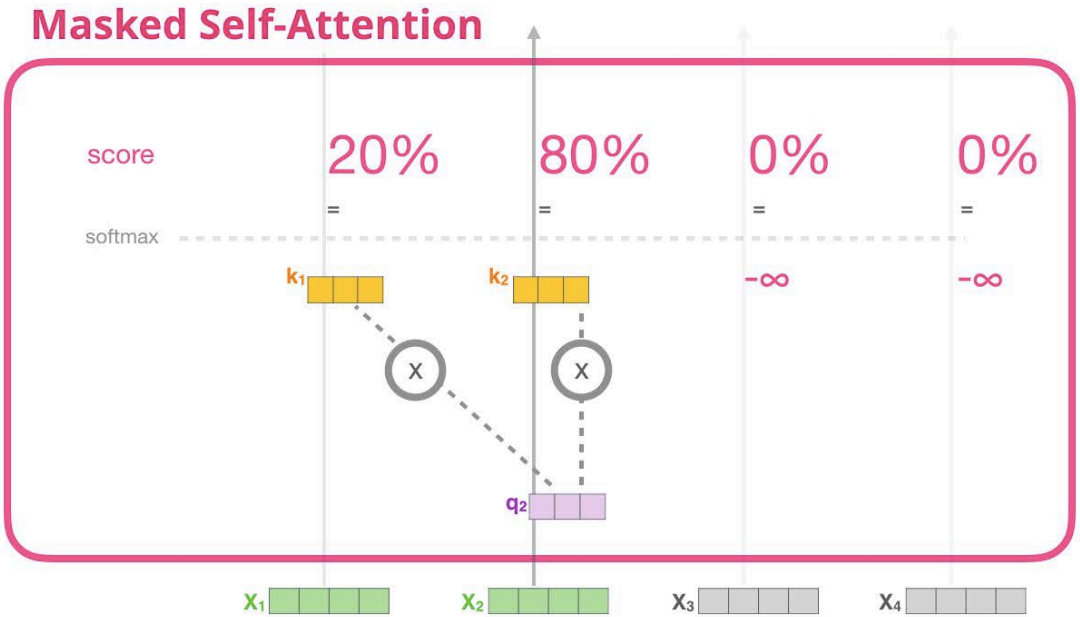
Presented to the next sublayer in the transformer block (the feed-forward neural network):



Part #2: The Illustrated Masked Self-Attention

Masked Self-Attention

Assuming the model only has two tokens as input and we're observing the second token. In this case, the last two tokens are masked. So the model interferes in the scoring step. It basically always scores the future tokens as 0 so the model can't peak to future words:



Part #2: The Illustrated Masked Self-Attention

Masked Self-Attention

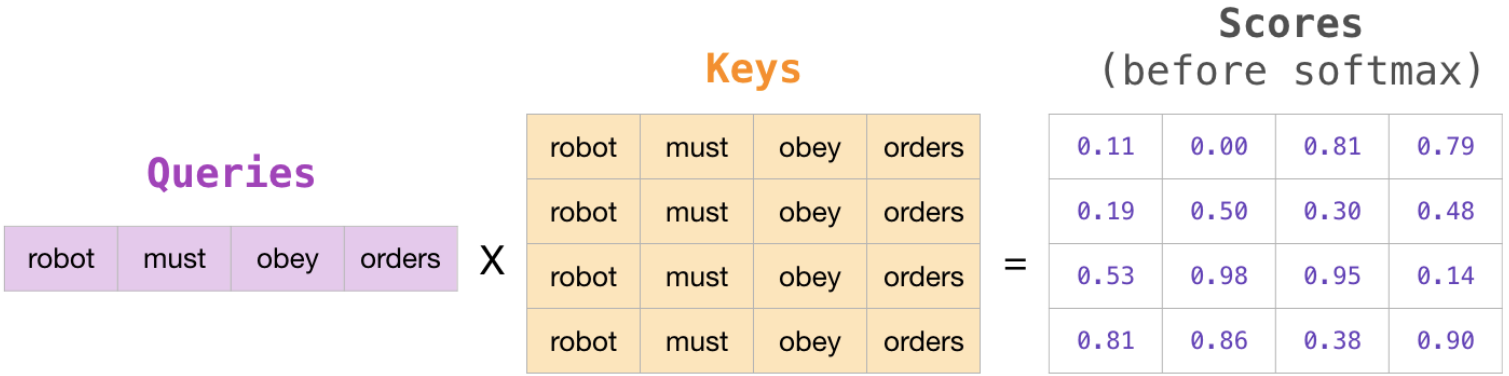
Implemented as a matrix called an attention mask.

Work in batches: assume a batch size of 4 that will process the entire sequence (with its four steps) as one batch.

	Features				Labels
	position: 1	2	3	4	
Example: 1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

Part #2: The Illustrated Masked Self-Attention

Matrix form: calculate the scores by multiplying a queries matrix by a keys matrix.



Part #2: The Illustrated Masked Self-Attention

Attention mask: after the multiplication, set the cells we want to mask to -infinity or a very large negative number (e.g. -1 billion in GPT2):

Scores
(before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Part #2: The Illustrated Masked Self-Attention

Then, applying softmax on each row produces the actual scores used for self-attention:

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax
(along rows)

Scores

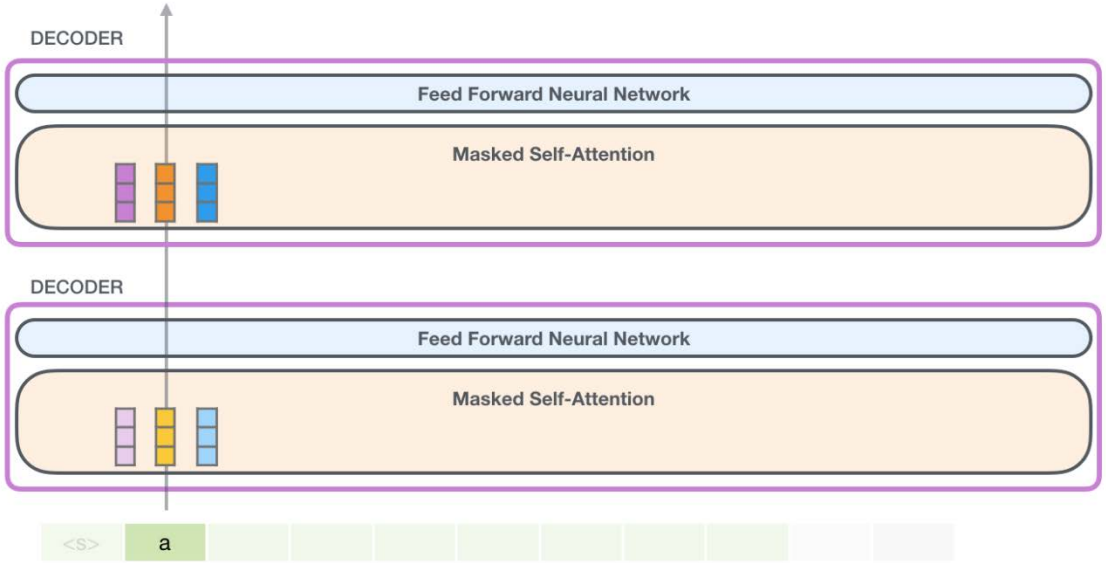
1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

- When the model processes the first example in the dataset (row #1), which contains only one word (“robot”), 100% of its attention will be on that word.
- When the model processes the second example in the dataset (row #2), which contains the words (“robot must”), when it processes the word “must”, 48% of its attention will be on “robot”, and 52% of its attention will be on “must”.
- And so on

Evaluation Time

Evaluation Time: Processing One Token at a Time

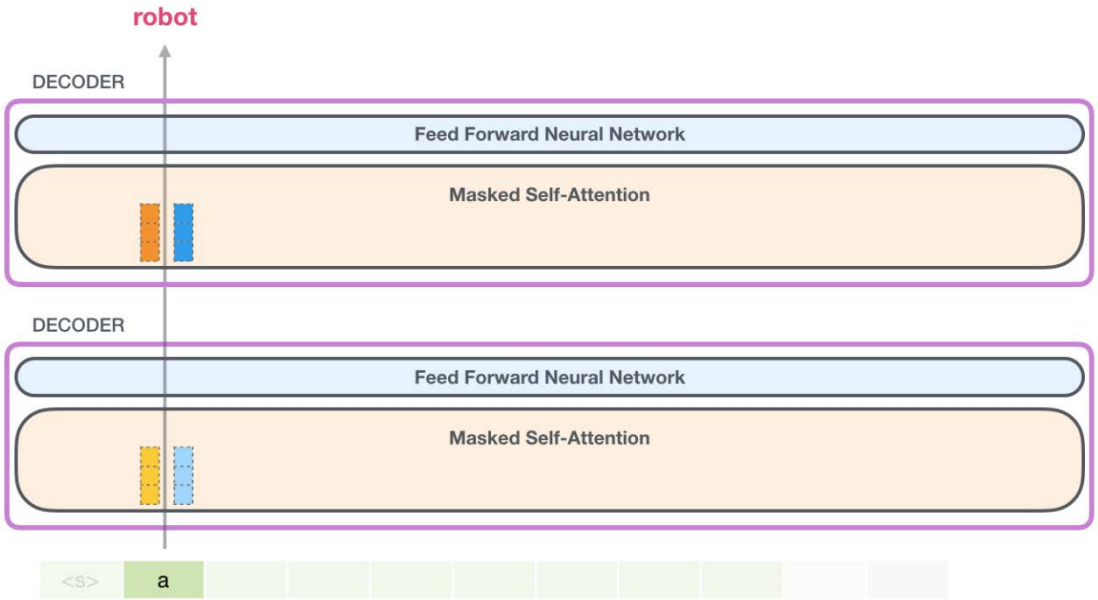
During evaluation, when the model is only adding one new word after each iteration, it would be **inefficient** to recalculate self-attention along earlier paths for tokens which have already been processed.



Evaluation Time

Evaluation Time: Processing One Token at a Time

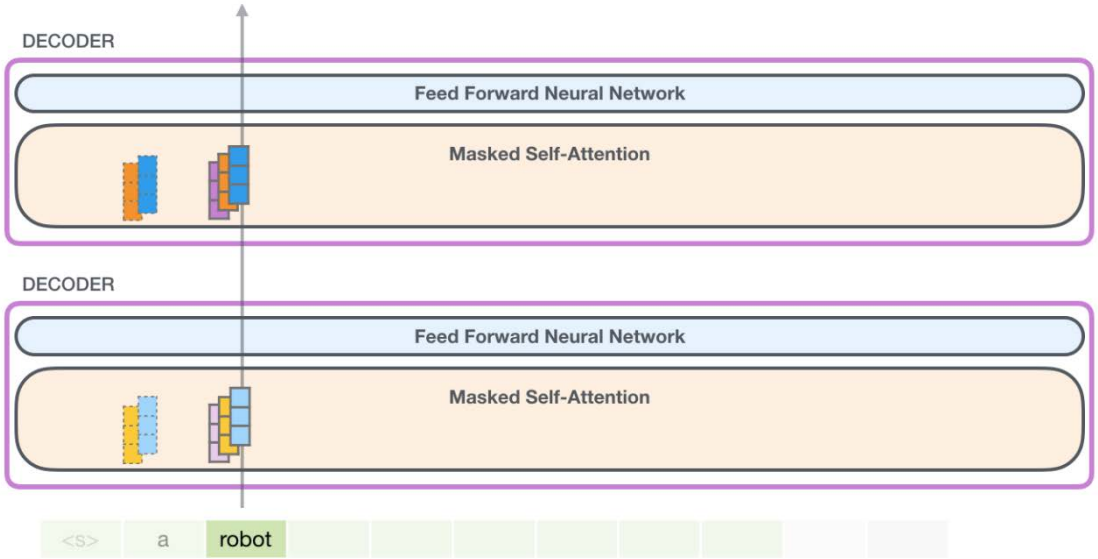
GPT-2 holds on to the key and value vectors of a token. Every self-attention layer holds on to its respective key and value vectors for that token:



Evaluation Time

Evaluation Time: Processing One Token at a Time

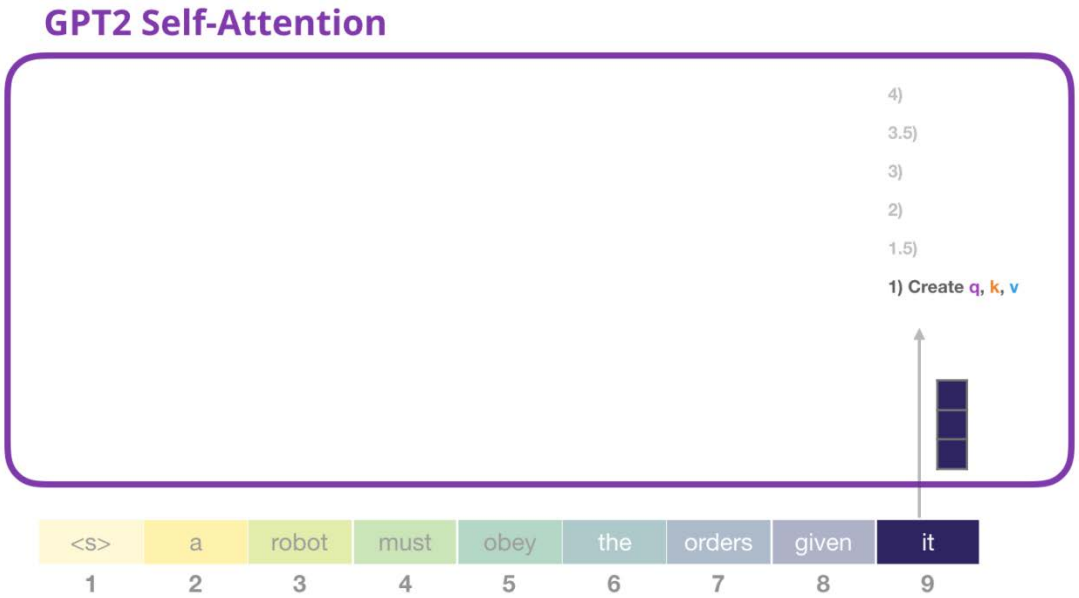
In the next iteration, when the model processes the word “robot”, it does not need to generate query, key, and value queries for the ‘a’ token. It just reuses the ones it saved from the first iteration:



GPT-2 Parameters

GPT-2 Self-attention: 1- Creating queries, keys, and values

When processing the word ‘it’, in the bottom block, the input for that token would be the embedding of “it” + the positional encoding for slot #9:

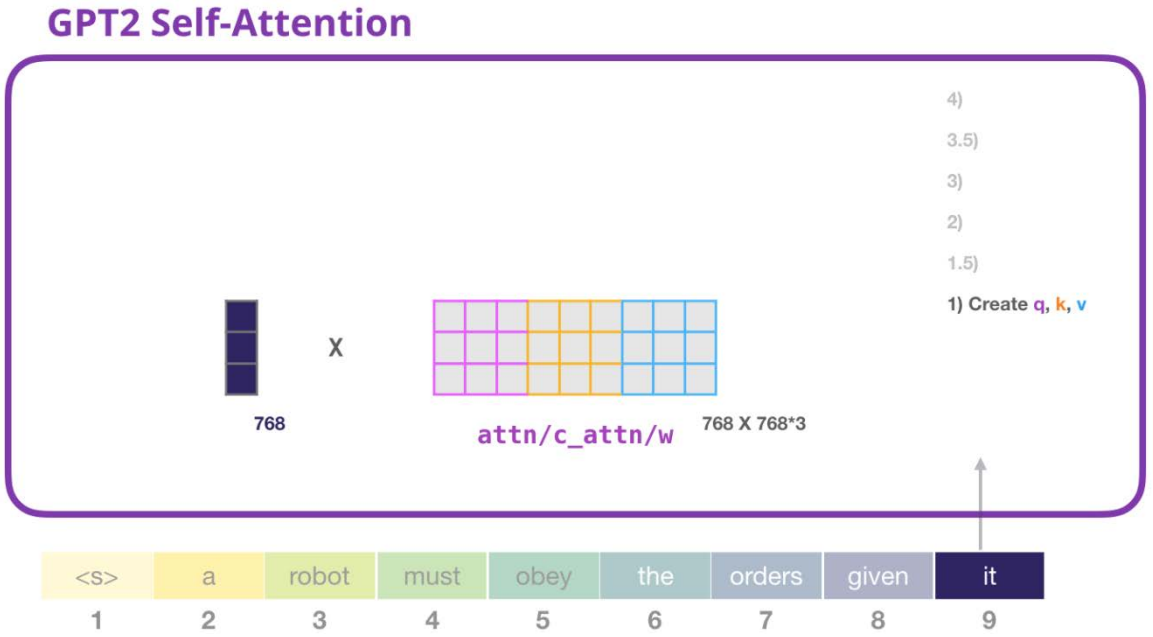


GPT-2 Parameters

Every block in a transformer has its own weights.

The first is the weight matrix that we use to create the queries, keys, and values.

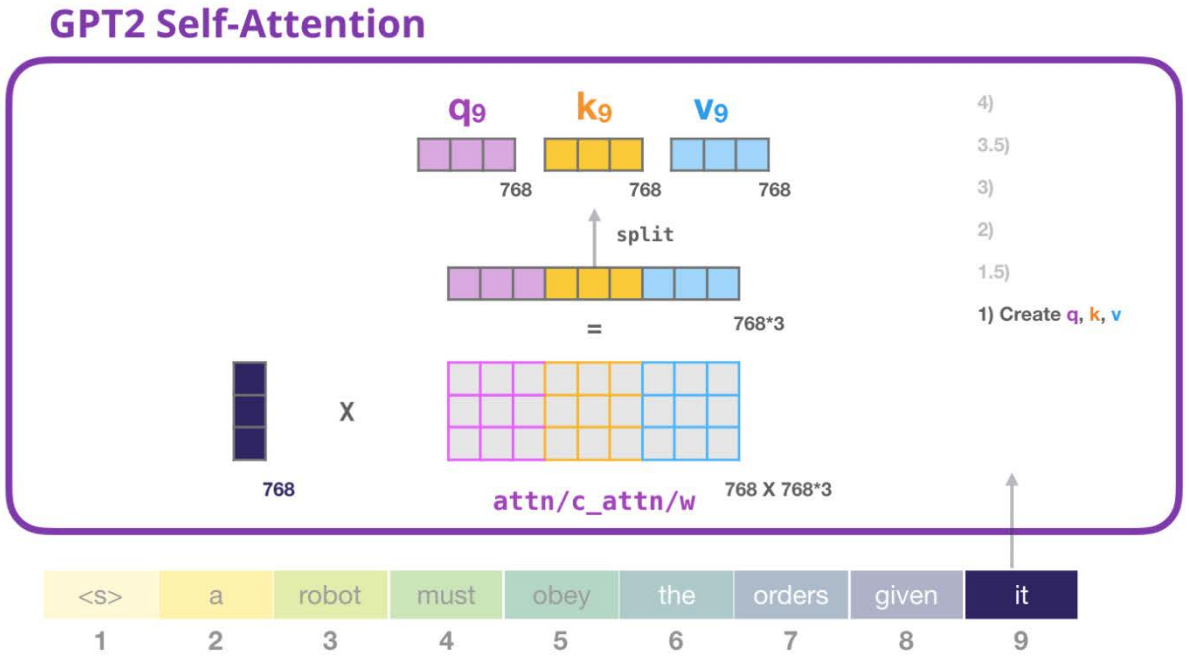
Self-attention multiplies its input by its weight matrix (and adds a bias vector).



GPT-2 Parameters

The multiplication results: a vector that's basically a concatenation of the query, key, and value vectors for the word "it".

Multiplying the input vector by the attention weights vector (and adding a bias vector afterwards) results in the key, value, and query vectors for this token.

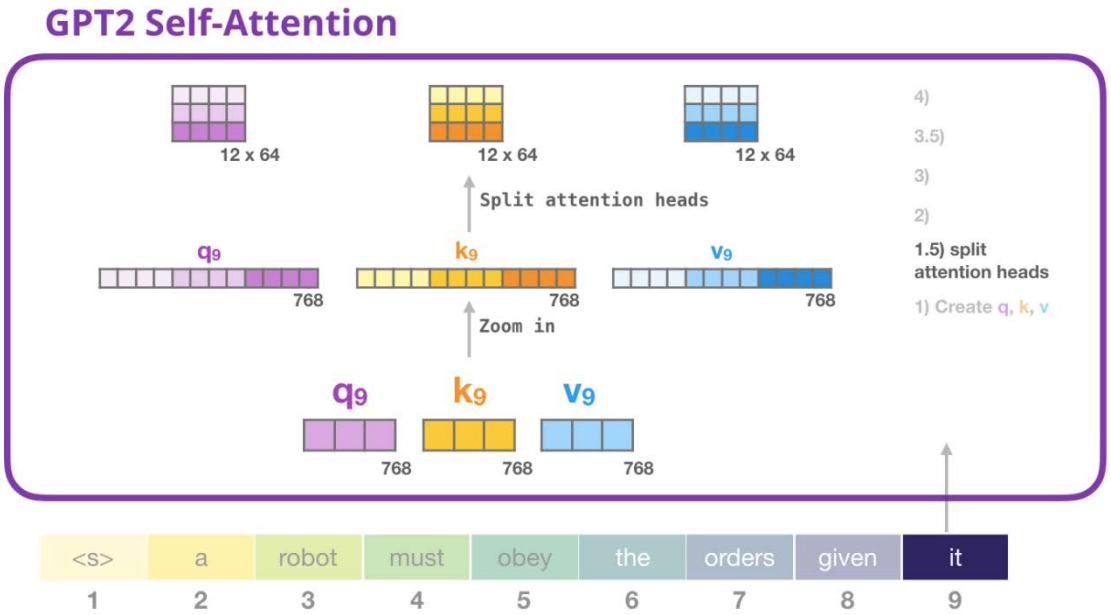


GPT-2 Parameters

GPT-2 Self-attention: 1.5- Splitting into attention heads

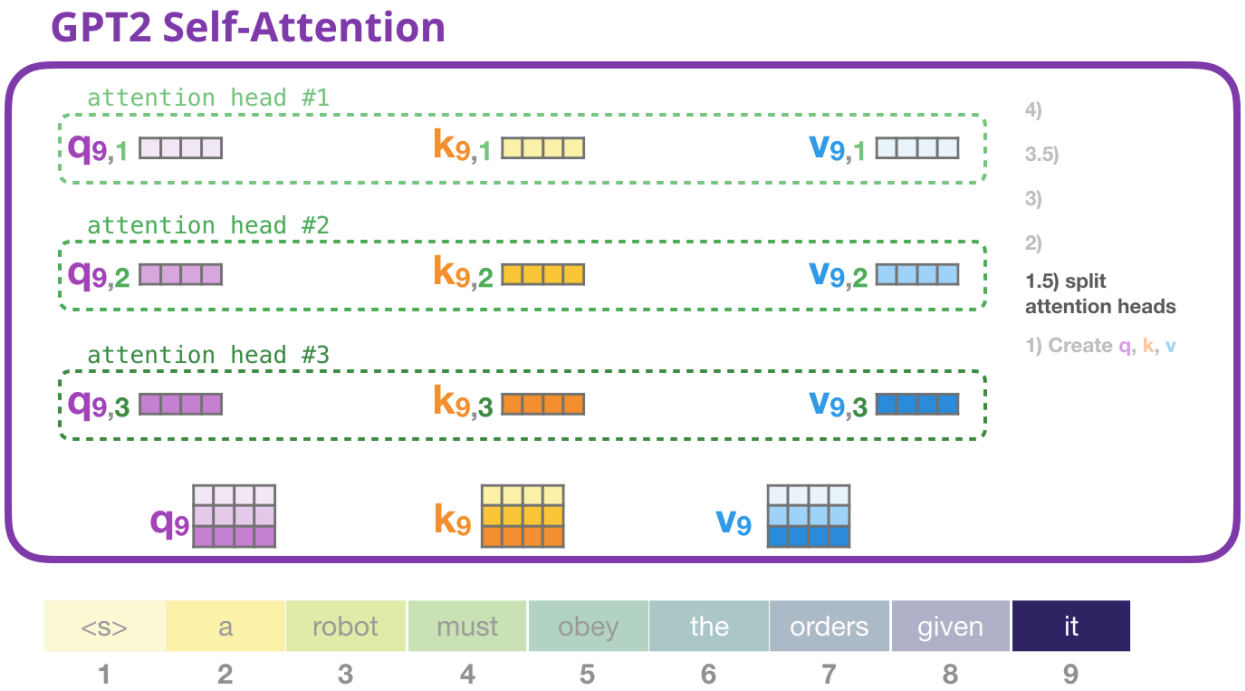
“Splitting” attention heads is simply reshaping the long vector into a matrix.

The small GPT2 has 12 attention heads, so that would be the first dimension of the reshaped matrix:



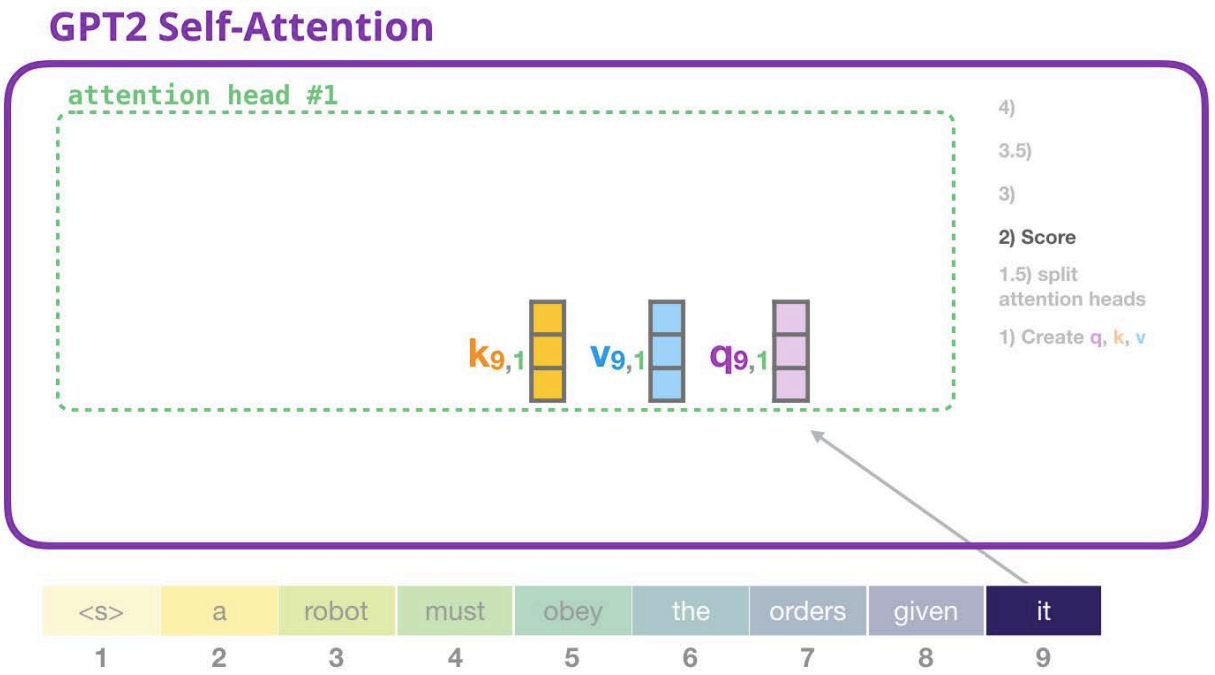
GPT-2 Parameters

Multiple attention-heads:



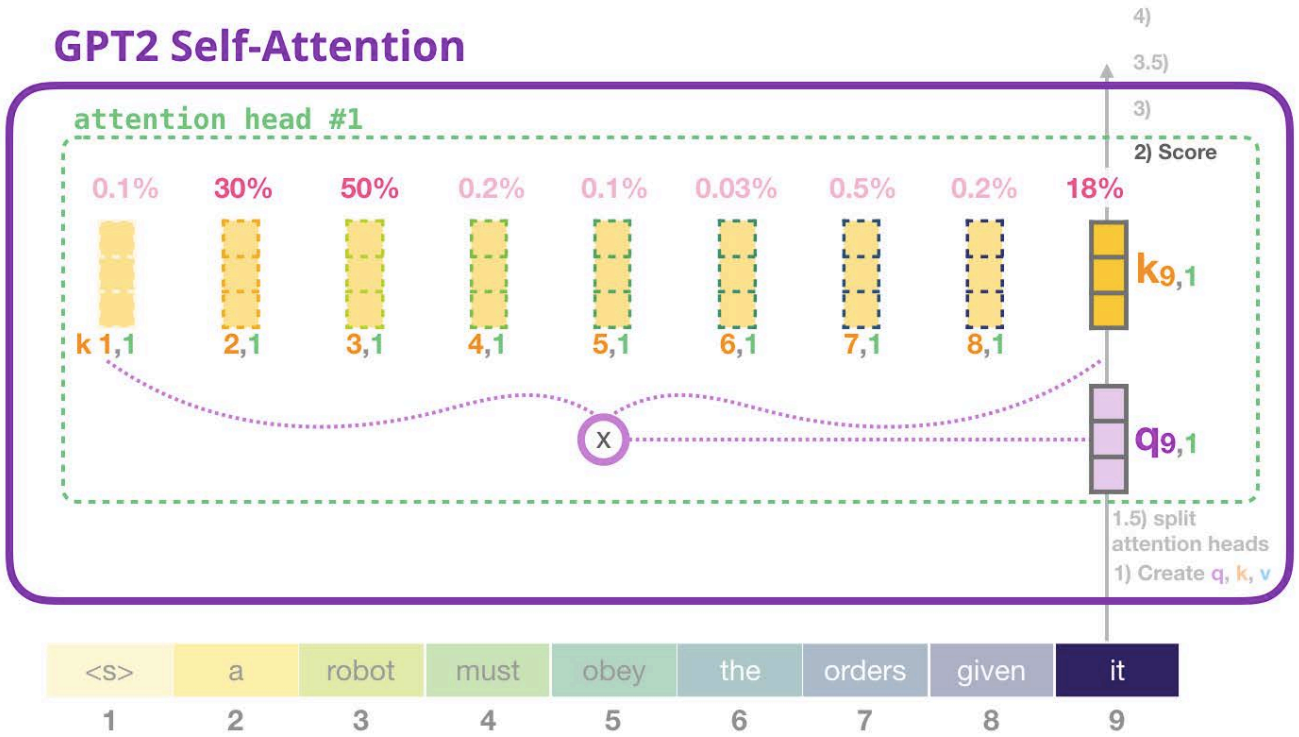
GPT-2 Parameters

GPT-2 Self-attention: 2- Scoring



GPT-2 Parameters

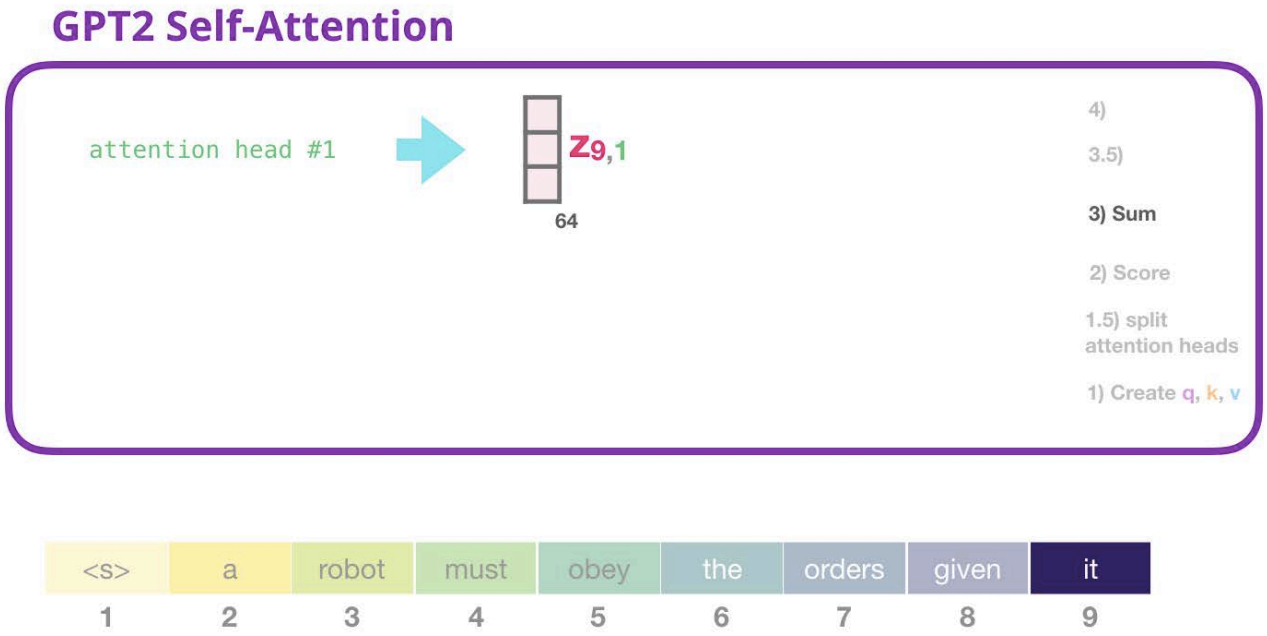
The token get scored against all of keys of the other tokens:



GPT-2 Parameters

GPT-2 Self-attention: 3- Sum

Multiply each value with its score, then sum them up, producing the result of self-attention for attention-head #1:

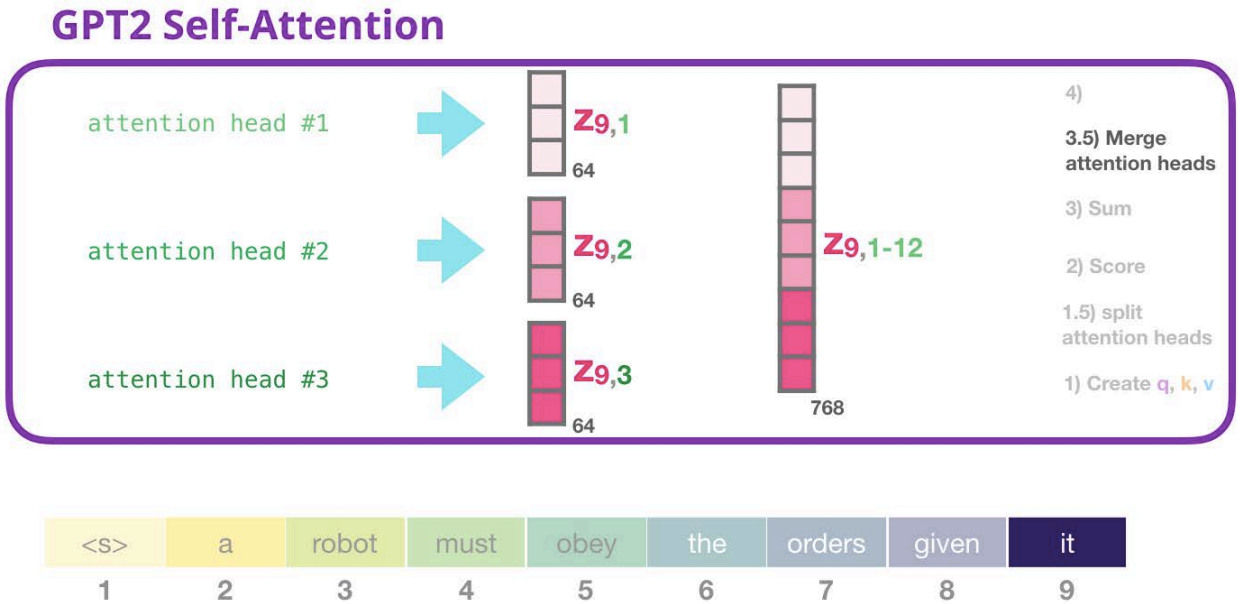


GPT-2 Parameters

GPT-2 Self-attention: 3.5- Merge attention heads

To deal with the various attention heads, firstly, concatenate them into one vector:

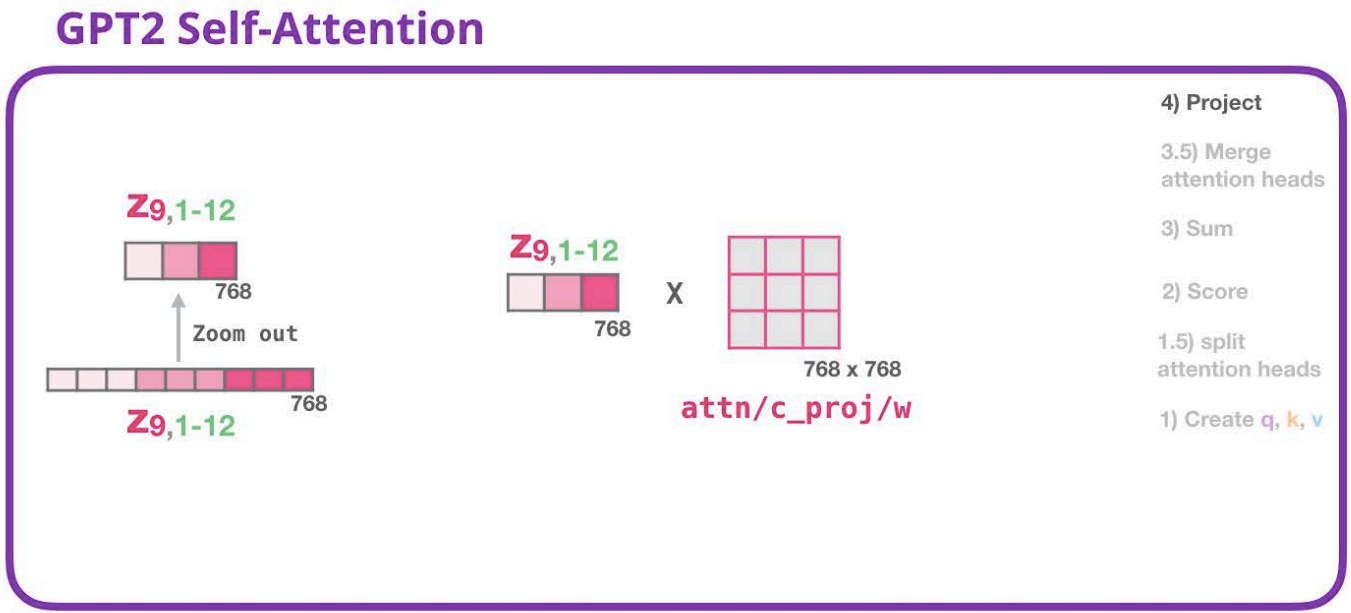
But the vector isn't ready to be sent to the next sublayer just yet. Need to first turn the hidden states into a homogenous representation.



GPT-2 Parameters

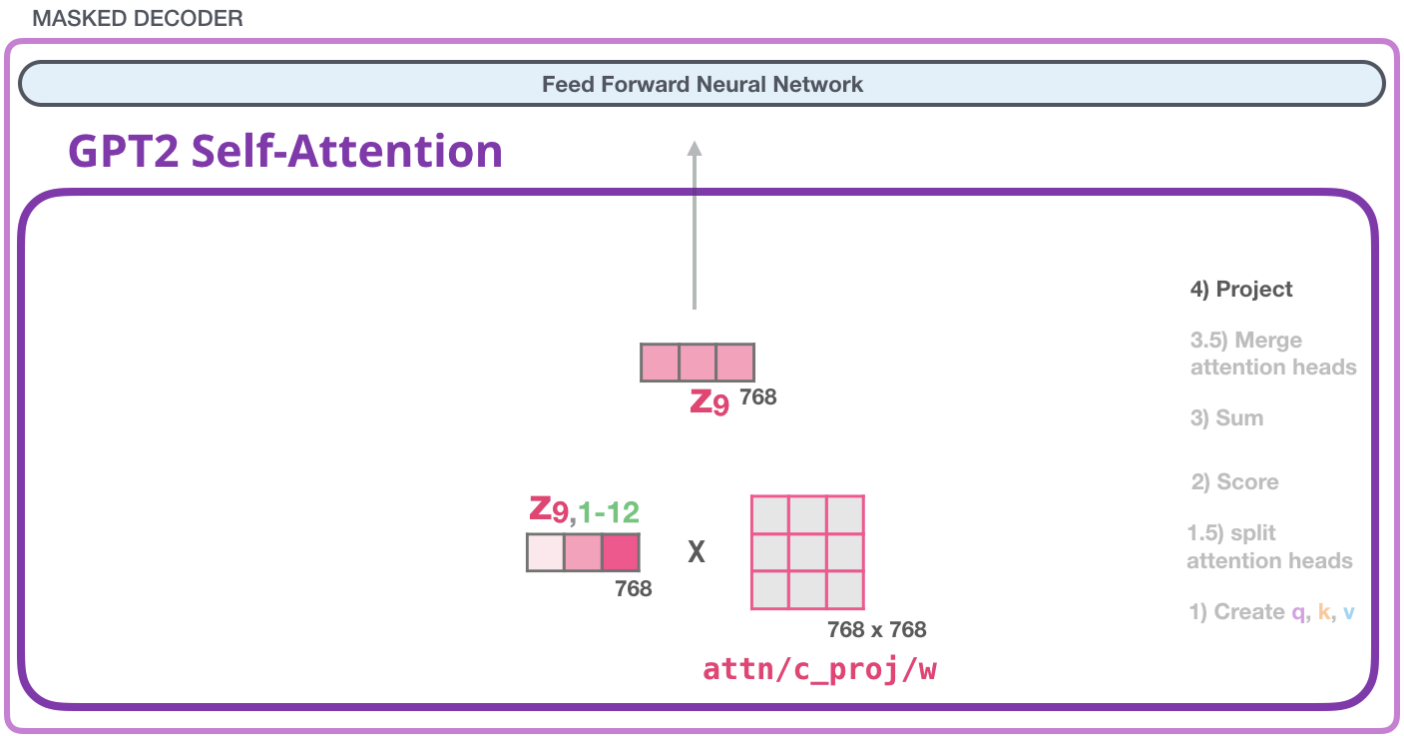
GPT-2 Self-attention: 4- Projecting

A large weight matrix that projects the results of the attention heads into the output vector of the self-attention sublayer:



GPT-2 Parameters

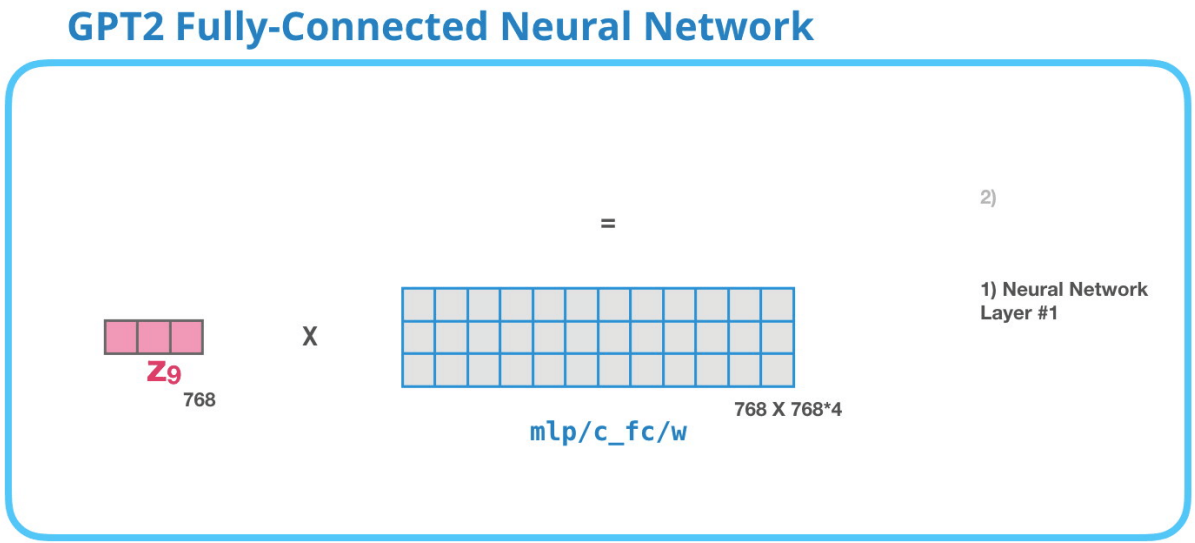
Produced the vector to send along to the next layer:



GPT-2 Parameters

GPT-2 Fully-Connected Neural Network: Layer #1

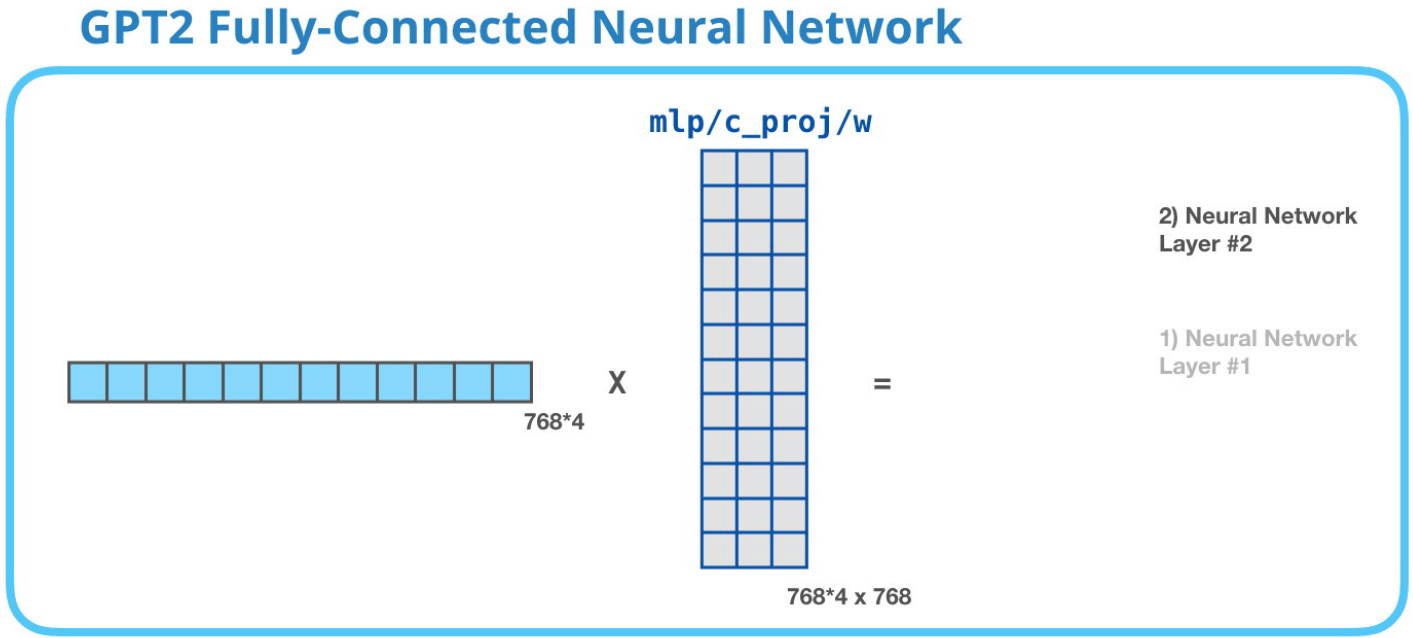
The first layer is four times the size of the model (GPT2 small is 768, this network would have 768*4 = 3072 units). Why four times? That's just the size the original transformer rolled with (model dimension was 512 and layer #1 in that model was 2048). This seems to give transformer models enough representational capacity to handle the tasks that have been thrown at them so far.



GPT-2 Parameters

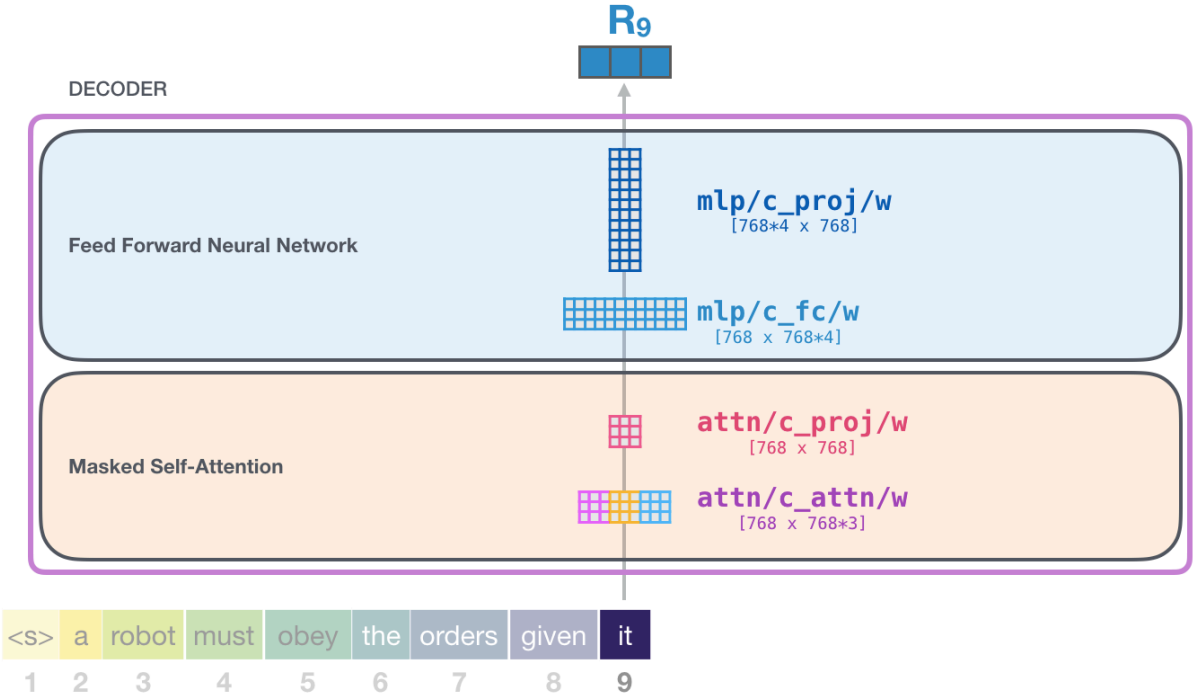
GPT-2 Fully-Connected Neural Network: Layer #2 - Projecting to model dimension

The second layer projects the result from the first layer back into model dimension (768 for the small GPT2). The result of this multiplication is the result of the transformer block for this token.



GPT-2 Parameters

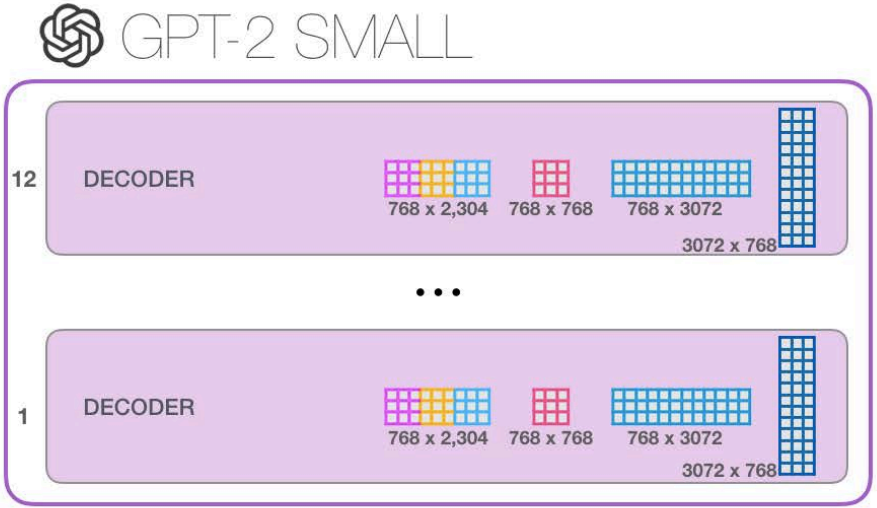
To recap, the input vector of “**it**” encounters these weight matrices:

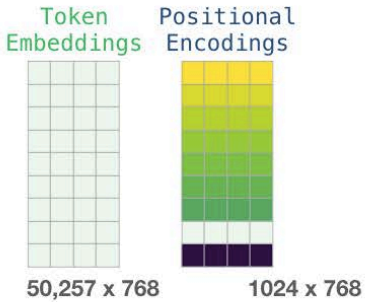


GPT-2 Parameters

Each block has its own set of these weights.

The model has only one token embedding matrix and one positional encoding matrix:





GPT-2 Parameters

To see all the parameters of the model:
Add up to 124M parameters instead of 117M for some reason. Not sure why, but that's how many of them seems to be in the published code.

				Dimensions		Parameters	
Single Transformer Block	Conv1d	**attn/c_attn**	w	768	2,304	1,769,472	
			b		2,304	2,304	
		attn/c_proj	w	768	768	589,824	
			b		768	768	
		mlp/c_fc	w	768	3,072	2,359,296	
			b		768	768	
		mlp/c_proj	w	3,072	768	2,359,296	
			b		768	768	
	Norm	**ln_1**	g		768	768	
			b		768	768	
		ln_2	g		768	768	
			b		768	768	
					Total	7,085,568	per block
					X 12 blocks	85,026,816	In all blocks
		Embeddings		50,257	768	38,597,376	
		Positional Encoding		1,024	768	786,432	
					Grand Total	**124,410,624**	

Part 3: Beyond Language Modeling

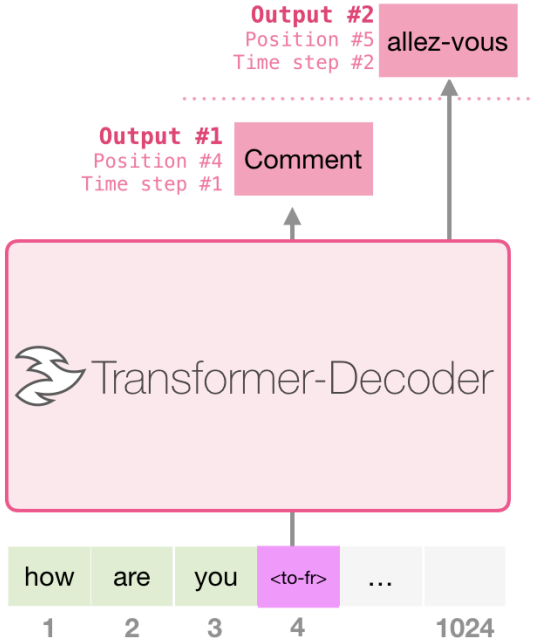
The decoder-only transformer keeps showing promise **beyond language modeling**.

Machine Translation

An encoder is not required to conduct translation. The same task can be addressed by a decoder-only transformer:

Training Dataset

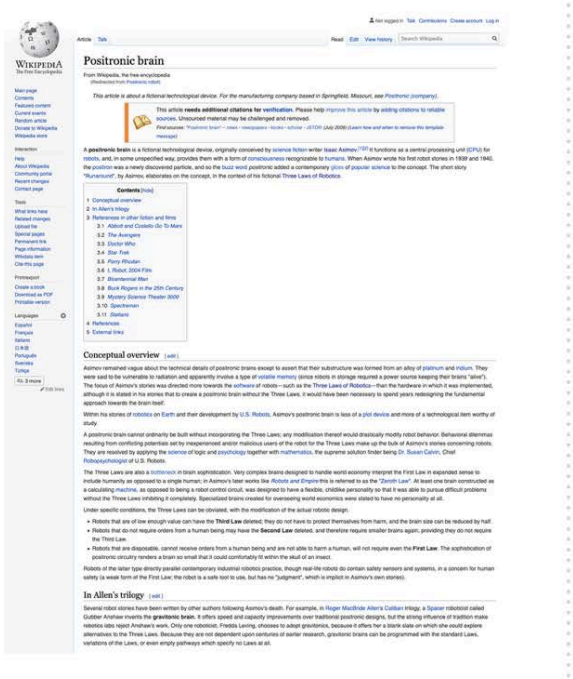
I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				



Part 3: Beyond Language Modeling

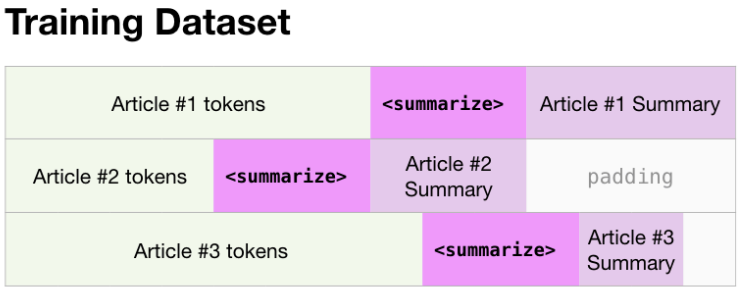
Summarization

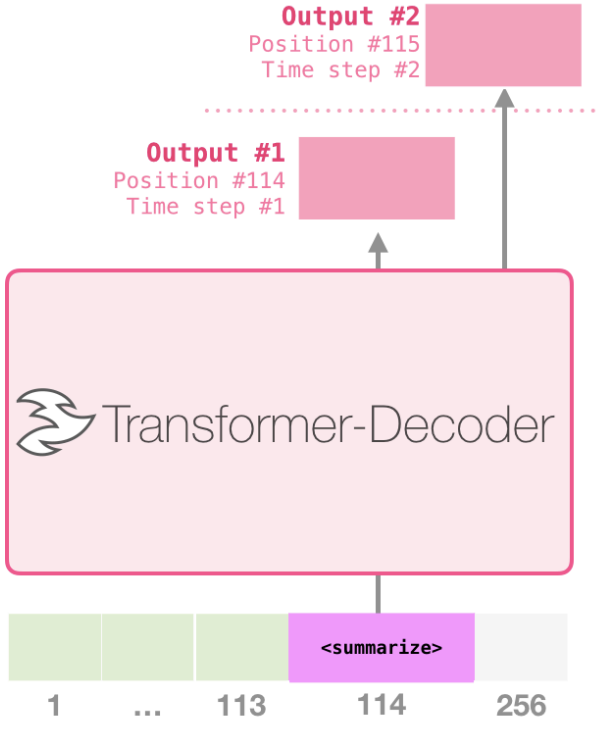
This is the task that the first decoder-only transformer was trained on. Namely, it was trained to read a wikipedia article (without the opening section before the table of contents), and to summarize it. The actual opening sections of the articles were used as the labels in the training datasest:



Part 3: Beyond Language Modeling

The paper trained the model against wikipedia articles, and thus the trained model was able to summarize articles:





Transfer Learning

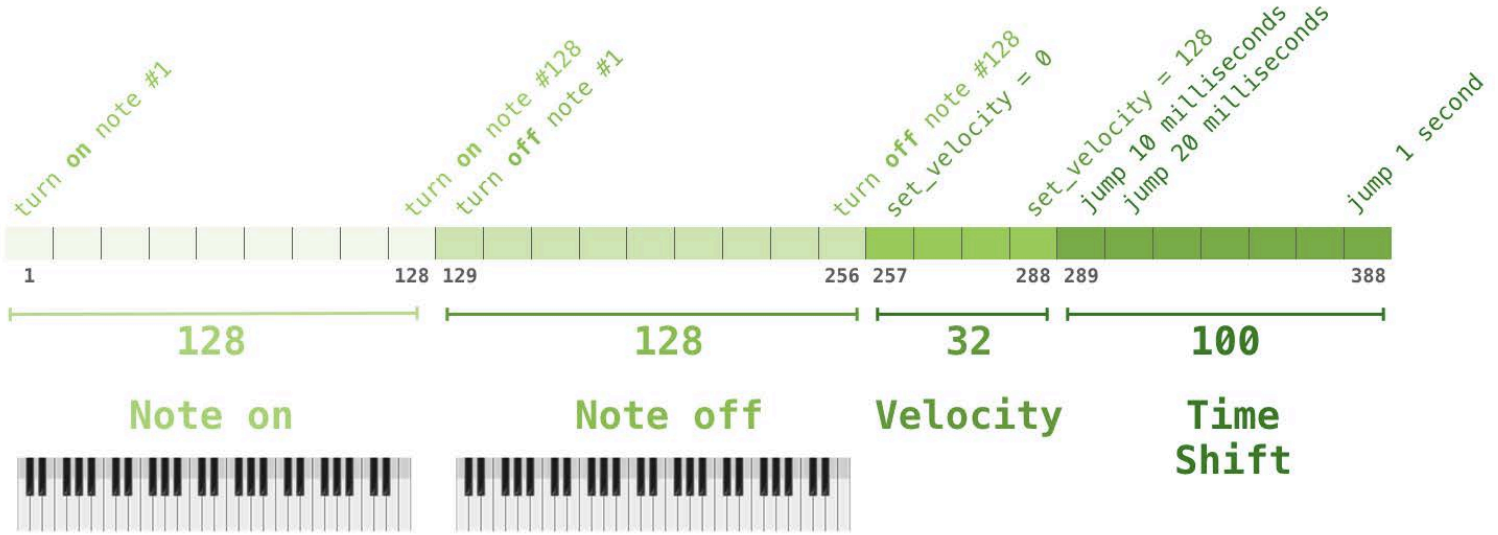
In Sample Efficient Text Summarization Using a Single Pre-Trained Transformer, a decoder-only transformer is first pre-trained on language modeling, then finetuned to do summarization. It turns out to achieve better results than a pre-trained encoder-decoder transformer in limited data settings.

The GPT2 paper also shows results of summarization after pre-training the model on language modeling.

Music Generation

- Music Transformer: uses a decoder-only transformer to generate music with expressive timing and dynamics.
- “Music Modeling”: let the model learn music in an unsupervised way, then have it sample outputs.
- How music is represented?

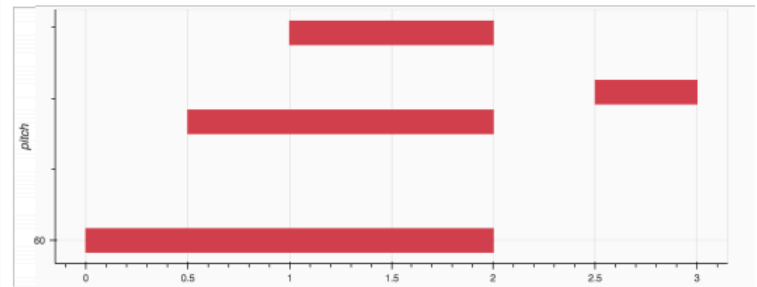
  -- With a musical performance (e.g., Piano), represent the notes and velocity – how hard the piano key is pressed.



Music Generation

A music performance is a series of these one-hot vectors.

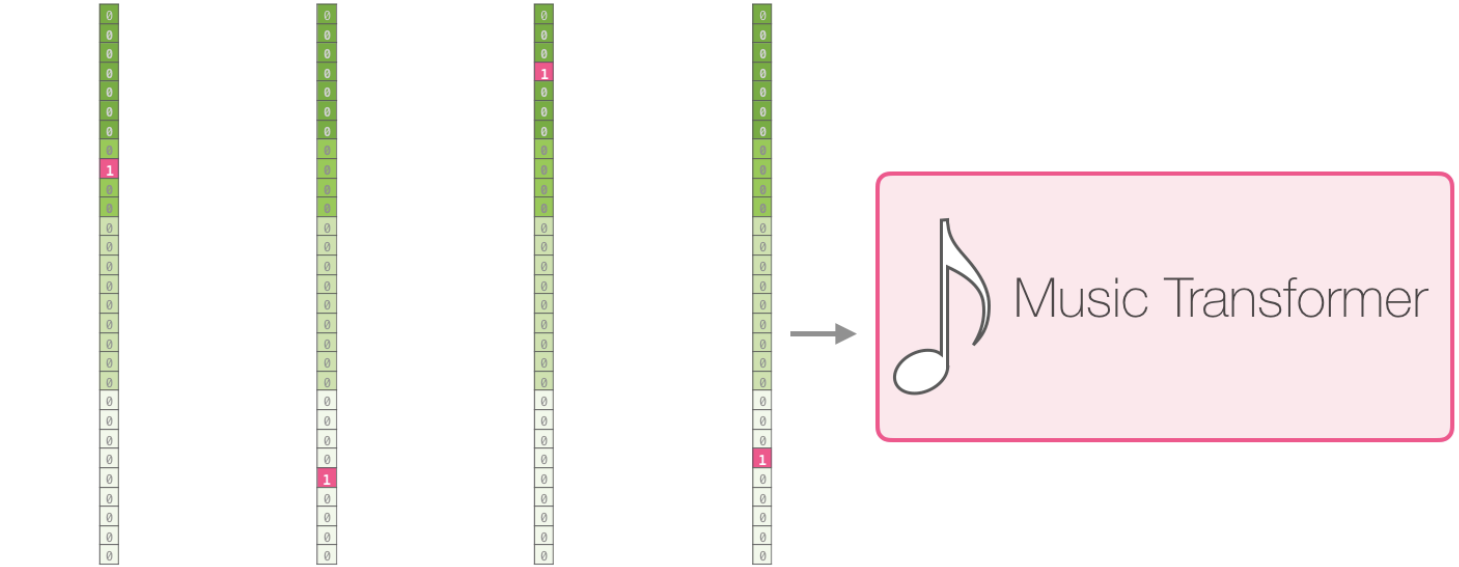
A midi file can be converted into such a format:




```
SET_VELOCITY<80>, NOTE_ON<60>
TIME_SHIFT<500>, NOTE_ON<64>
TIME_SHIFT<500>, NOTE_ON<67>
TIME_SHIFT<1000>, NOTE_OFF<60>, NOTE_OFF<64>,
NOTE_OFF<67>
TIME_SHIFT<500>, SET_VELOCITY<100>, NOTE_ON<65>
TIME_SHIFT<500>, NOTE_OFF<65>
```

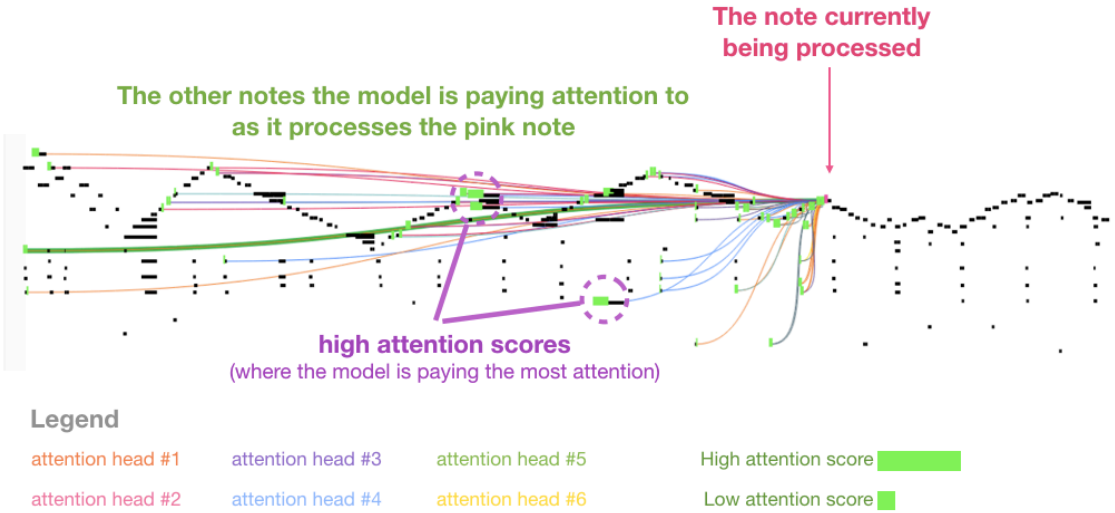
Music Generation

The one-hot vector representation for this input sequence would look like this:



Music Generation

A visual showcases self-attention in the Music Transformer.

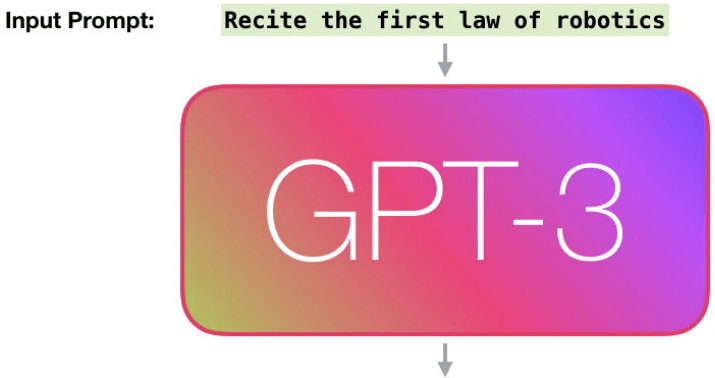


"Figure: This piece has a recurring triangular contour. The query is at one of the latter peaks and it attends to all of the previous high notes on the peak, all the way to beginning of the piece." ... "[The] figure shows a query (the source of all the attention lines) and previous memories being attended to (the notes that are receiving more softmax probabiliy is highlighted in). The coloring of the attention lines correspond to different heads and the width to the weight of the softmax probability."

Music Generation

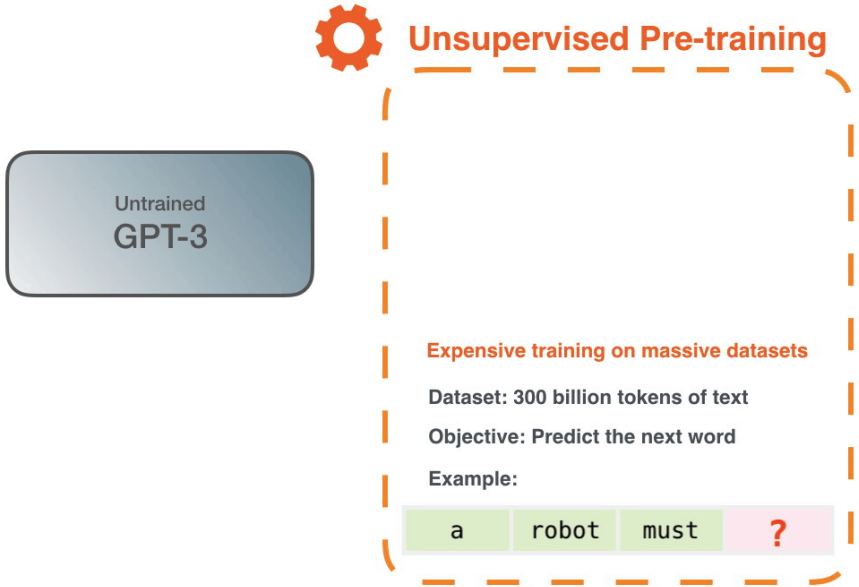
How GPT3 Works - Visualizations and Animations

GPT3: A trained language model generates text.



How GPT3 Works - Visualizations and Animations

Training based on lots of text. Cost 355 GPU years and cost $4.6m.



How GPT3 Works - Visualizations and Animations

The dataset of 300 billion tokens of text is used to generate training examples for the model.

Text: Second Law of Robotics: A robot must obey the orders given it by human beings

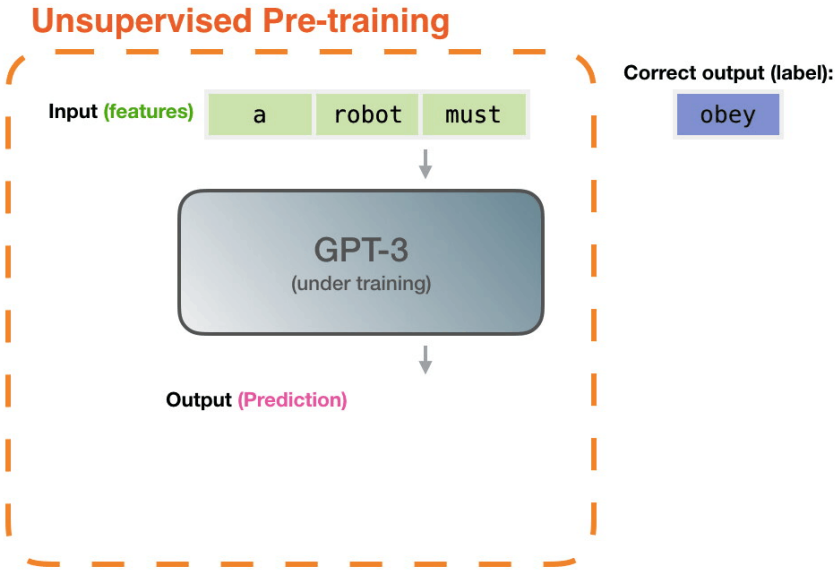
Generated training examples

Example #	Input (features)	Correct output (labels)
1	Second law of robotics :	a
2	Second law of robotics : a	robot
3	Second law of robotics : a robot	must
...		

How GPT3 Works - Visualizations and Animations

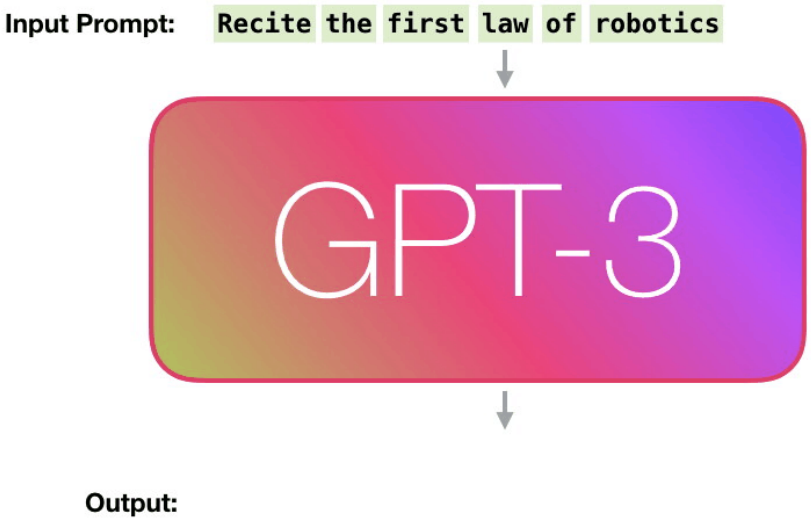
Training: Calculate the error in its prediction and update the model so next time it makes a better prediction.

Repeat millions of times.



How GPT3 Works - Visualizations and Animations

GPT3 actually generates output one token at a time.

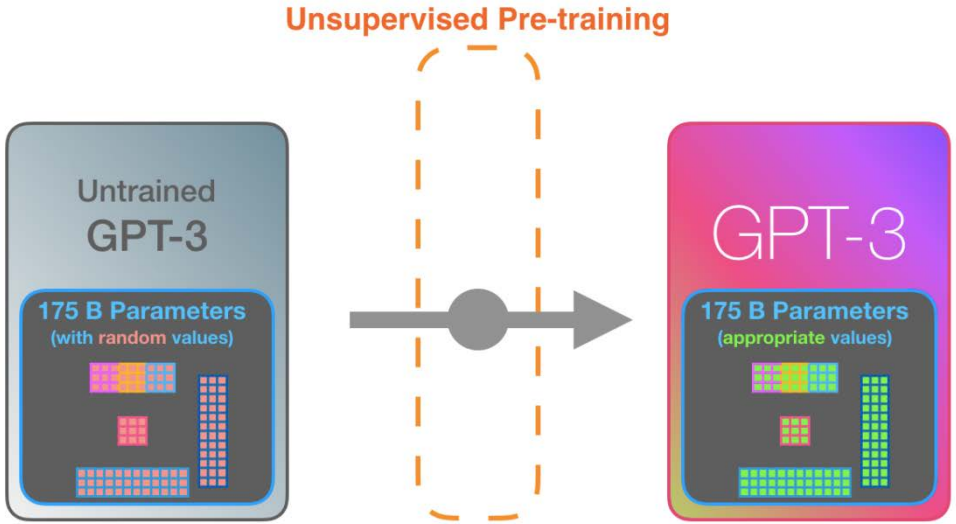


How GPT3 Works - Visualizations and Animations

GPT3 is **MASSIVE**, with 175 billion parameters.

The untrained model starts with random parameters.

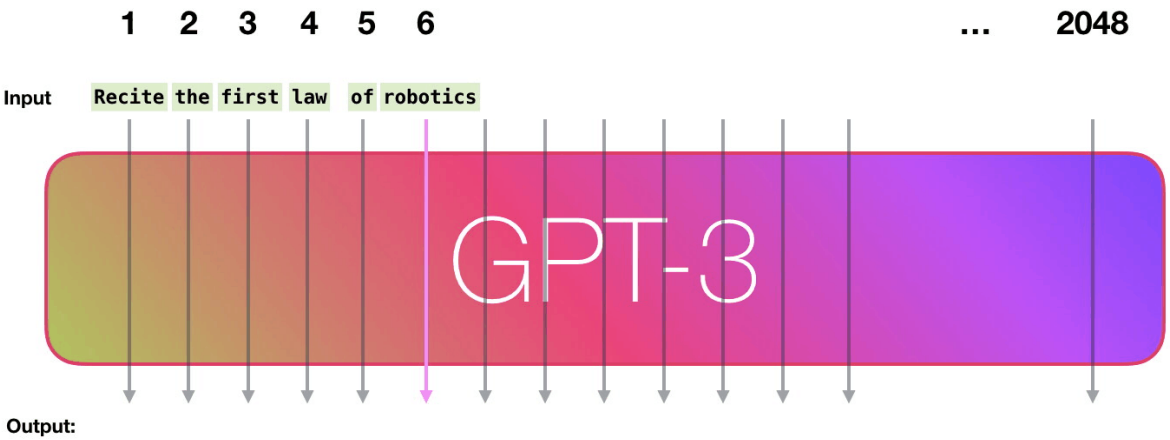
Training finds values that lead to better predictions.



How GPT3 Works - Visualizations and Animations

Prediction is mostly a lot of matrix multiplication.

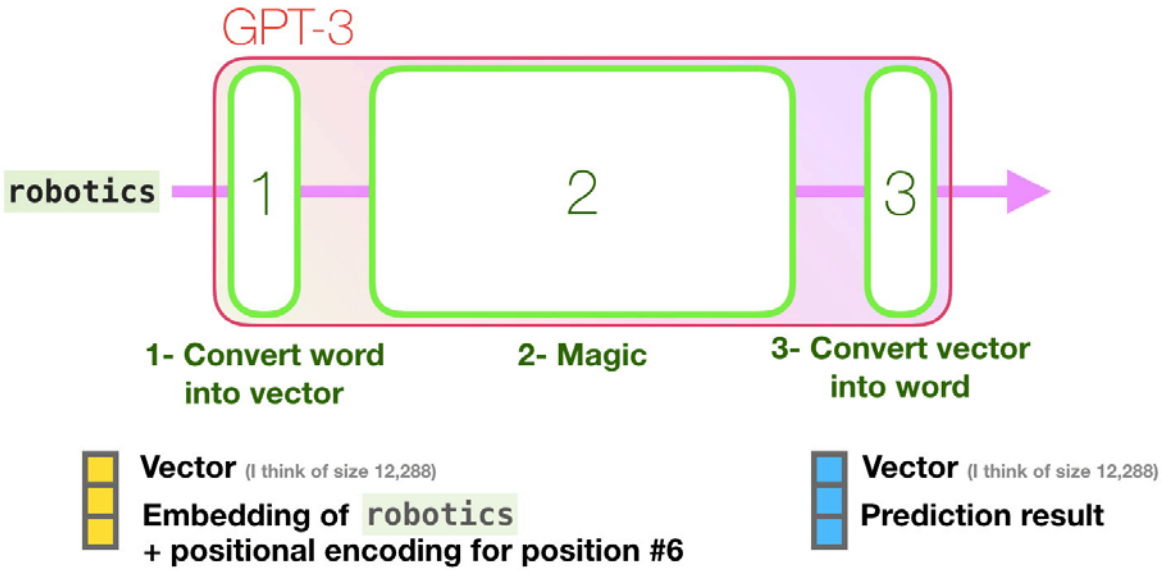
GPT3 is 2048 tokens wide. That is its “**context window**”.



How GPT3 Works - Visualizations and Animations

High-level steps:

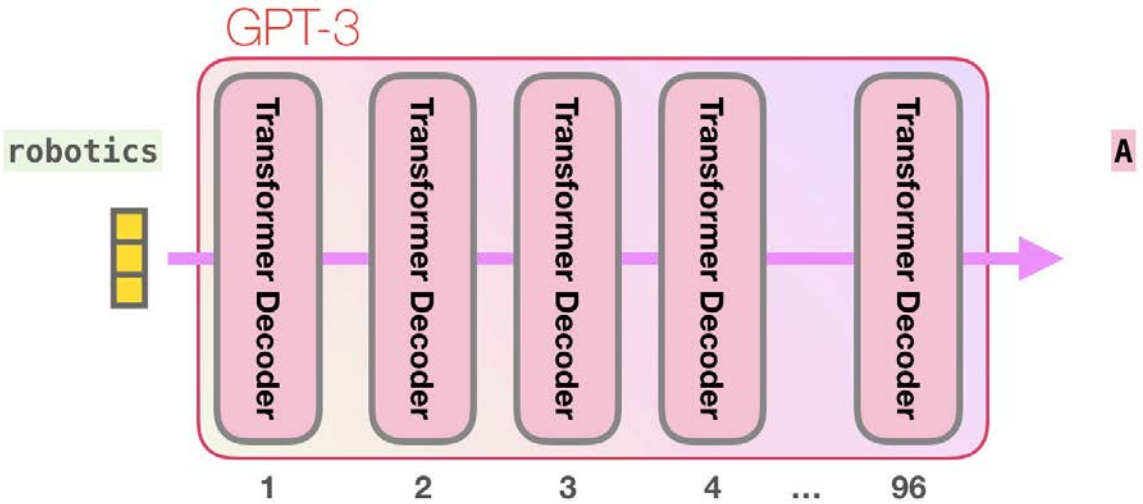
- Convert the word to a vector (list of numbers) representing the word
- Compute prediction
- Convert resulting vector to word



How GPT3 Works - Visualizations and Animations

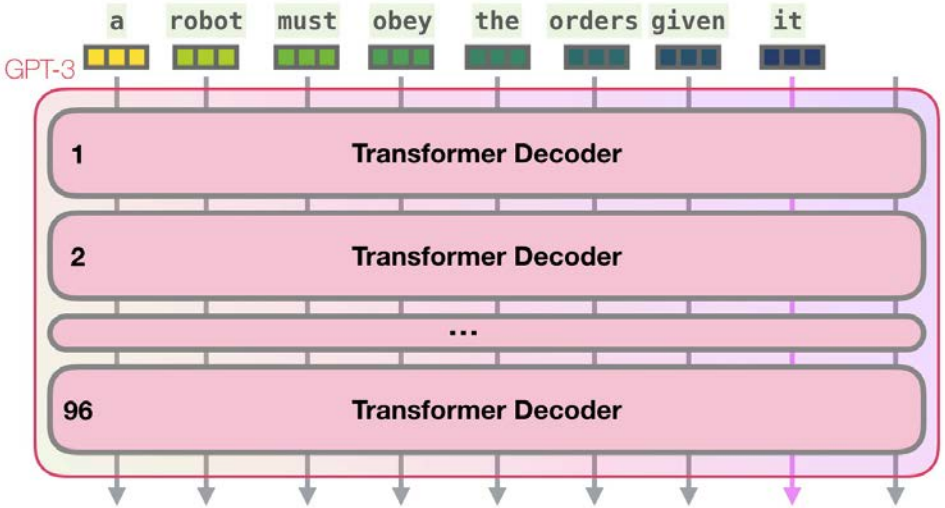
GPT3 “**depth**”: 96 transformer decoder layers.

Each of these decoder layers has its own 1.8B parameters.



How GPT3 Works - Visualizations and Animations

Every token flows through the entire layer stack.



How GPT3 Works - Visualizations and Animations

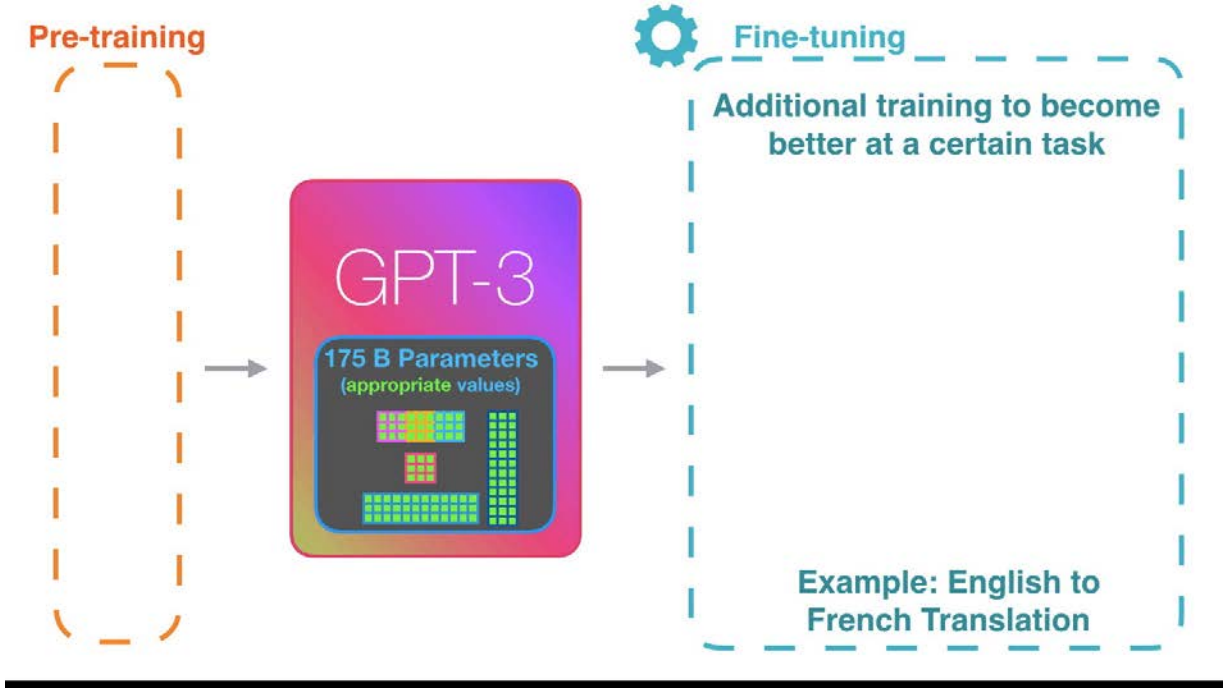
Code generation example:

Input prompt (in green), and a couple of examples of description=>code.

The react code would be generated like the pink tokens.

How GPT3 Works - Visualizations and Animations

Fine-tuning actually updates the model's weights to make the model better at a certain task.



问题及讨论

Q&A