Transformers

March 13, 2024

1 Introduction to Transformers

- Used for modeling long dependencies between input sequence elements
- Supports parallel processing of sequence as compared to RNN (e.g. LSTM)
- Allows processing multiple modalities
- `(e.g., images, videos, text and speech) using similar processing blocks`
- Typically, pre-trained using pretext tasks on largescale (unlabeled) datasets
- Demonstrates excellent scalability to very large networks and huge datasets.

1.1 1.1 History

In 2017, researchers at Google published a paper that proposed a novel neural network architecture for sequence modeling. Dubbed the Transformer, this architecture outperformed recurrent neural networks (RNNs) on machine translation tasks, both in terms of translation quality and training cost.

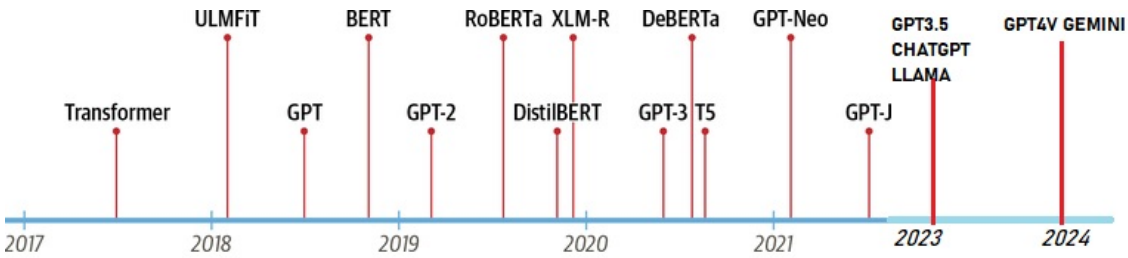
A. Vaswani et al., "Attention Is All You Need", (2017). This title was so catchy that no less than 50 follow-up papers have included "all you need" in their titles!

In parallel, an effective transfer learning method called ULMFiT showed that training long short-term memory (LSTM) networks on a very large and diverse corpus could produce state-of-the-art text classifiers with little labeled data.

J. Howard and S. Ruder, "Universal Language Model Fine-Tuning for Text Classification", (2018).

These advances were the catalysts for two of today's most well-known transformers: - the Generative Pretrained Transformer (GPT) and - Bidirectional Encoder Representations from Transformers (BERT).

By combining the Transformer architecture with unsupervised learning, these models removed the need to train task-specific architectures from scratch and broke almost every benchmark in NLP by a significant margin.



2 1.2 A Tour of Transformer Applications with HUGGINGFACE

2.1 Text Classification

[1]:
```
!pip install transformers
!pip install ipywidgets
!jupyter nbextension enable --py widgetsnbextension
```

```
Requirement already satisfied: transformers in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (4.37.2)
Requirement already satisfied: requests in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(2.31.0)
Requirement already satisfied: numpy>=1.17 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(1.23.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(0.20.3)
Requirement already satisfied: pyyaml>=5.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers) (6.0.1)
Requirement already satisfied: packaging>=20.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers) (21.3)
Requirement already satisfied: safetensors>=0.4.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers) (0.4.2)
Requirement already satisfied: tqdm>=4.27 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(4.65.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(0.15.2)
Requirement already satisfied: filelock in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(3.13.1)
Requirement already satisfied: regex!=2019.12.17 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from transformers)
(2023.12.25)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from huggingface-
```

```
hub<1.0,>=0.19.3->transformers) (4.9.0)
Requirement already satisfied: fsspec>=2023.5.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from huggingface-
hub<1.0,>=0.19.3->transformers) (2024.2.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
packaging>=20.0->transformers) (3.0.9)
Requirement already satisfied: colorama in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
tqdm>=4.27->transformers) (0.4.5)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
requests->transformers) (3.6)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
requests->transformers) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
requests->transformers) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
requests->transformers) (2024.2.2)
Requirement already satisfied: ipywidgets in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (8.1.2)
Requirement already satisfied: jupyterlab-widgets~=3.0.10 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from ipywidgets) (3.0.10)
Requirement already satisfied: traitlets>=4.3.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from ipywidgets) (5.3.0)
Requirement already satisfied: widgetsnbextension~=4.0.10 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from ipywidgets) (4.0.10)
Requirement already satisfied: ipython>=6.1.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from ipywidgets) (8.4.0)
Requirement already satisfied: comm>=0.1.3 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from ipywidgets) (0.2.1)
Requirement already satisfied: backcall in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: stack-data in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.3.0)
Requirement already satisfied: colorama in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.4.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (3.0.30)
Requirement already satisfied: setuptools>=18.5 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
```

```
ipython>=6.1.0->ipywidgets) (63.3.0)
Requirement already satisfied: pickleshare in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.7.5)
Requirement already satisfied: pygments>=2.4.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (2.12.0)
Requirement already satisfied: matplotlib-inline in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.1.3)
Requirement already satisfied: jedi>=0.16 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (0.18.1)
Requirement already satisfied: decorator in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: wcwidth in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from prompt-
toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=6.1.0->ipywidgets) (0.2.5)
Requirement already satisfied: executing in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from stack-
data->ipython>=6.1.0->ipywidgets) (0.9.1)
Requirement already satisfied: asttokens in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from stack-
data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from stack-
data->ipython>=6.1.0->ipywidgets) (0.2.2)
Requirement already satisfied: six in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from asttokens->stack-
data->ipython>=6.1.0->ipywidgets) (1.16.0)

Enabling notebook extension jupyter-js-widgets/extension…
      - Validating: ok
```

[2]:
```python
from transformers import pipeline
classifer=pipeline('text-classification')
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-
sst-2-english and revision af0f99b (https://huggingface.co/distilbert-base-
uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in production is
not recommended.
```

[3]:
```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

[4]:
```
import pandas as pd
outputs = classifer(text)
pd.DataFrame(outputs)
```

[4]:

	label	score
0	NEGATIVE	0.901545

2.2 Named Entity Recognition

[5]:
```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

```
No model was supplied, defaulted to dbmdz/bert-large-cased-finetuned-
conll03-english and revision f2482bf (https://huggingface.co/dbmdz/bert-large-
cased-finetuned-conll03-english).
Using a pipeline without specifying a model name and revision in production is
not recommended.
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-
conll03-english were not used when initializing BertForTokenClassification:
['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForTokenClassification from
the checkpoint of a model that you expect to be exactly identical (initializing
a BertForSequenceClassification model from a BertForSequenceClassification
model).
```

[5]:

	entity_group	score	word	start	end
0	ORG	0.879010	Amazon	5	11
1	MISC	0.990859	Optimus Prime	36	49
2	LOC	0.999755	Germany	90	97
3	MISC	0.556569	Mega	208	212
4	PER	0.590256	##tron	212	216
5	ORG	0.669692	Decept	253	259
6	MISC	0.498350	##icons	259	264
7	MISC	0.775361	Megatron	350	358
8	MISC	0.987854	Optimus Prime	367	380

```
9          PER  0.812096      Bumblebee    502  511
```

2.3 Question Answering

[6]:
```python
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline

QAtokenizer = AutoTokenizer.from_pretrained("SRDdev/QABERT-small")

QAmodel = AutoModelForQuestionAnswering.from_pretrained("SRDdev/QABERT-small")

ask = pipeline("question-answering", model= QAmodel , tokenizer = QAtokenizer)

result = ask(question="What does the customer want?", context=text)

print(f"Answer: '{result['answer']}'")
```

```
Answer: 'an exchange of Megatron'
```

2.4 Summarization

[7]:
```python
!pip install sentencepiece
```

```
Requirement already satisfied: sentencepiece in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (0.1.99)
```

[8]:
```python
from transformers import pipeline, AutoTokenizer, AutoModelForSeq2SeqLM

summarizer = pipeline("summarization", model="Abijith/Billsum-text-summarizer-
t5-small")

outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
print(outputs[0]['summary_text'])
```

```
Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my
horror that I had been sent an action figure of Mega
```

2.5 Translation

[9]:
```python
from transformers import pipeline


translator = pipeline("text2text-generation", model="barghavani/
 ↪English_to_Hindi")
outputs = translator(text, clean_up_tokenization_spaces=True, min_length=50)
print(outputs[0]['generated_text'])
```

```
C:\Users\Admin\miniconda3\envs\ML_\lib\site-
packages\transformers\models\marian\tokenization_marian.py:197: UserWarning:
```

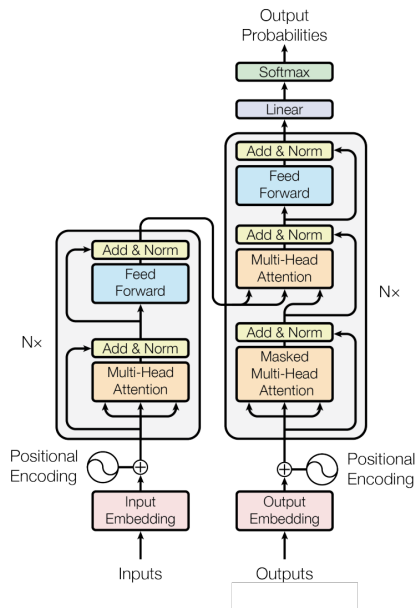
```
Recommended: pip install sacremoses.
  warnings.warn("Recommended: pip install sacremoses.")
```

,
. ,
.
. ,
. .

3 1.3 Architecture

BERT

Encoder

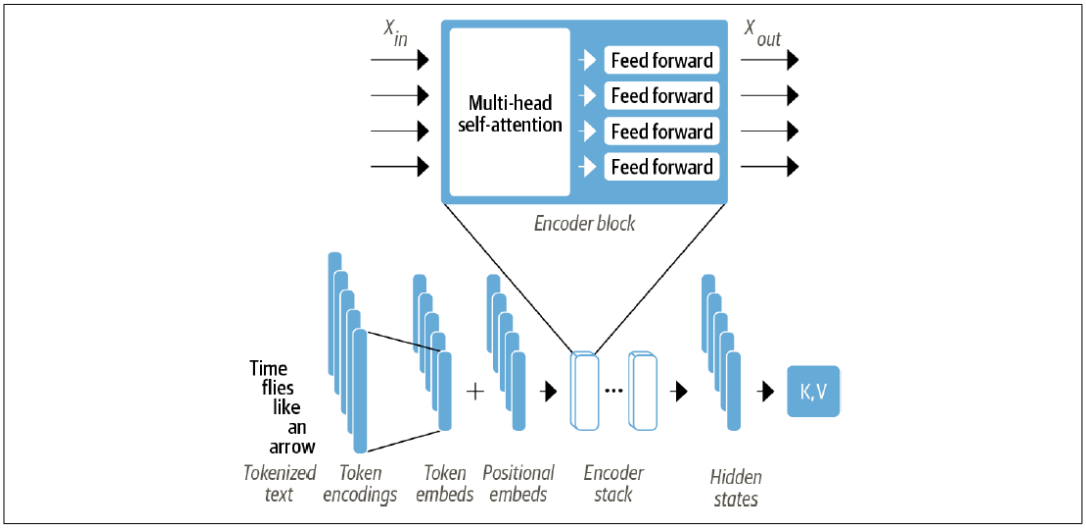


GPT

Decoder

4 Encoder

The transformer's encoder consists of many encoder layers stacked next to each other. As illustrated in Figure below, each encoder layer receives a sequence of embeddings and feeds them through the following sublayers: - A multi-head self-attention layer - A fully connected feed-forward layer that is applied to each input embedding



The output embedding for the word "apple" will be updated to be more "company-like" and less "fruit-like" if the words "keynote" or "phone" are close to it.

4.0.1 Self Attention

The main idea behind self-attention is that instead of using a fixed embedding for each token, we can use the whole sequence to compute a weighted average of each embedding.

Another way to formulate this is to say that given a sequence of token embeddings $x_1, \dots, x_n$, , self-attention produces a sequence of new embeddings $y_1, \dots, y_n$ where each y_i is a linear combination of all the x_j:

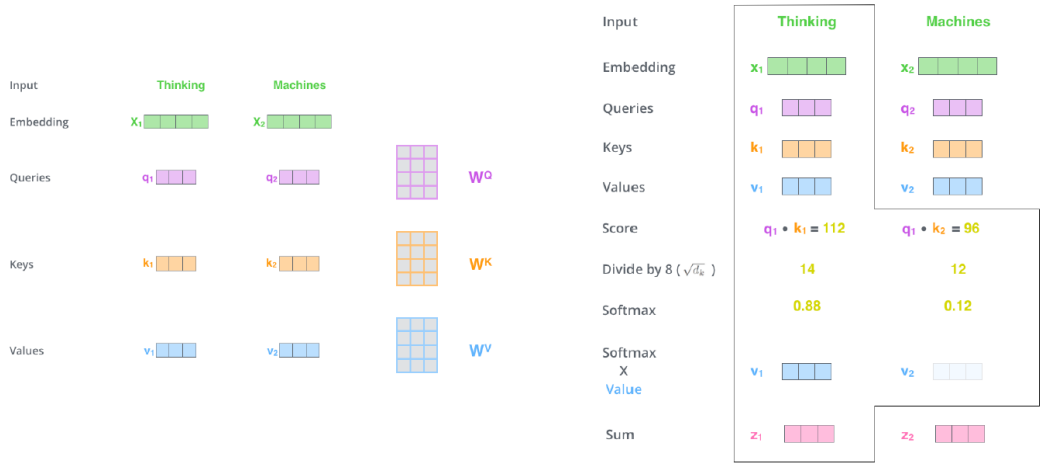
$$y_i = \sum_{j=1}^{n} w_{ij} x_j$$

The coefficients w_{ij} are called attention weights and are normalized so that $\sum_j w_{ij} = 1$.

But why? Consider the word flies. Any embedding algorithm that produces the embedding of words in the vocabulary produces a single d-dimensional embedding for the word flies. Consider following two sentences: - time flies like an arrow - fruit flies like a banana

Self-attention tries to discrimate between the two occurences of flies by looking at the context where the word was used. The resulting embeddings are called *contextualized embeddings*.

How? — Scaled dot product attention Given a sequence of d dimensional token embeddings $x_1, \dots, x_n$, , self-attention produces a sequence of new embeddings $y_1, \dots, y_n$ by using three steps: - Project each token embedding into three vectors called query, key, and value $q_i = W^Q x_i$, $k_i = W^K x_i$, $v_i = W^V x_i$ - for each q_i, compute attention scores and attention distribution: $\alpha_{i,j} = Softmax(\frac{q_i \cdot k_j}{\sqrt{d}})$ - compute the weighted sum: $y_i = \sum_{j=1}^{n} \alpha_{i,j} v_j$



Self-attention: matrix notations

$X \in \mathbb{R}^{n \times d_1}$

Note: the notations we use here are following the original paper
(= the transpose of the matrices in previous notations)

$Q = XW^Q \quad K = XW^K \quad V = XW^V$

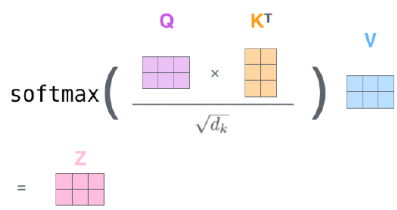
$W^Q \in \mathbb{R}^{d_1 \times d_q}, W^K \in \mathbb{R}^{d_1 \times d_k}, W^V \in \mathbb{R}^{d_1 \times d_v}$

$n \times d_q$ $\quad$ $d_k \times n$

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

$n \times d_v$

Q: What is this softmax operation?



Visualizing Attention with bertviz

[10]:
```
!pip install bertviz
```

```
Requirement already satisfied: bertviz in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (1.4.0)
Requirement already satisfied: requests in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (2.31.0)
Requirement already satisfied: boto3 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (1.34.41)
Requirement already satisfied: torch>=1.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (2.2.0)
Requirement already satisfied: tqdm in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (4.65.0)
Requirement already satisfied: sentencepiece in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (0.1.99)
Requirement already satisfied: regex in
```

```
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (2023.12.25)
Requirement already satisfied: transformers>=2.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from bertviz) (4.37.2)
Requirement already satisfied: filelock in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(3.13.1)
Requirement already satisfied: typing-extensions>=4.8.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(4.9.0)
Requirement already satisfied: sympy in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(1.12)
Requirement already satisfied: networkx in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(3.2.1)
Requirement already satisfied: jinja2 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(3.1.2)
Requirement already satisfied: fsspec in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from torch>=1.0->bertviz)
(2024.2.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (0.15.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (0.20.3)
Requirement already satisfied: numpy>=1.17 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (1.23.1)
Requirement already satisfied: packaging>=20.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (21.3)
Requirement already satisfied: safetensors>=0.4.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (0.4.2)
Requirement already satisfied: pyyaml>=5.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
transformers>=2.0->bertviz) (6.0.1)
Requirement already satisfied: colorama in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from tqdm->bertviz)
(0.4.5)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from boto3->bertviz)
(1.0.1)
Requirement already satisfied: s3transfer<0.11.0,>=0.10.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from boto3->bertviz)
(0.10.0)
```

```
Requirement already satisfied: botocore<1.35.0,>=1.34.41 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from boto3->bertviz)
(1.34.41)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from requests->bertviz)
(2024.2.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from requests->bertviz)
(2.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from requests->bertviz)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from requests->bertviz)
(3.6)
Collecting urllib3<3,>=1.21.1
  Using cached urllib3-2.0.7-py3-none-any.whl (124 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
botocore<1.35.0,>=1.34.41->boto3->bertviz) (2.8.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
packaging>=20.0->transformers>=2.0->bertviz) (3.0.9)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
jinja2->torch>=1.0->bertviz) (2.1.1)
Requirement already satisfied: mpmath>=0.19 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from
sympy->torch>=1.0->bertviz) (1.3.0)
Requirement already satisfied: six>=1.5 in
c:\users\admin\miniconda3\envs\ml_\lib\site-packages (from python-
dateutil<3.0.0,>=2.1->botocore<1.35.0,>=1.34.41->boto3->bertviz) (1.16.0)
Installing collected packages: urllib3
  Attempting uninstall: urllib3
    Found existing installation: urllib3 2.1.0
    Uninstalling urllib3-2.1.0:
      Successfully uninstalled urllib3-2.1.0
Successfully installed urllib3-2.0.7
```

[11]:
```python
from transformers import AutoTokenizer
from bertviz.transformers_neuron_view import BertModel
from bertviz.neuron_view import show

model_ckpt = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = BertModel.from_pretrained(model_ckpt)
text = "time flies like an arrow"
```

```
show(model, "bert", tokenizer, text, display_mode="light", layer=0, head=8)
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

C:\Users\Admin\miniconda3\envs\ML_\lib\site-
packages\huggingface_hub\file_download.py:149: UserWarning: `huggingface_hub`
cache-system uses symlinks by default to efficiently store duplicated files but
your machine does not support them in
C:\Users\Admin\.cache\huggingface\hub\models--bert-base-uncased. Caching files
will still work but in a degraded version that might require more space on your
disk. This warning can be disabled by setting the
`HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see
https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to
run Python as an administrator. In order to see activate developer mode, see
this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-
your-device-for-development
  warnings.warn(message)

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Multi-headed attention As seen in the previous slide, the self-attention layer applies three independent linear transformations to each embedding to generate the query, key, and value vectors.

It also turns out to be beneficial to have multiple sets of linear projections, each one representing a so-called attention head.

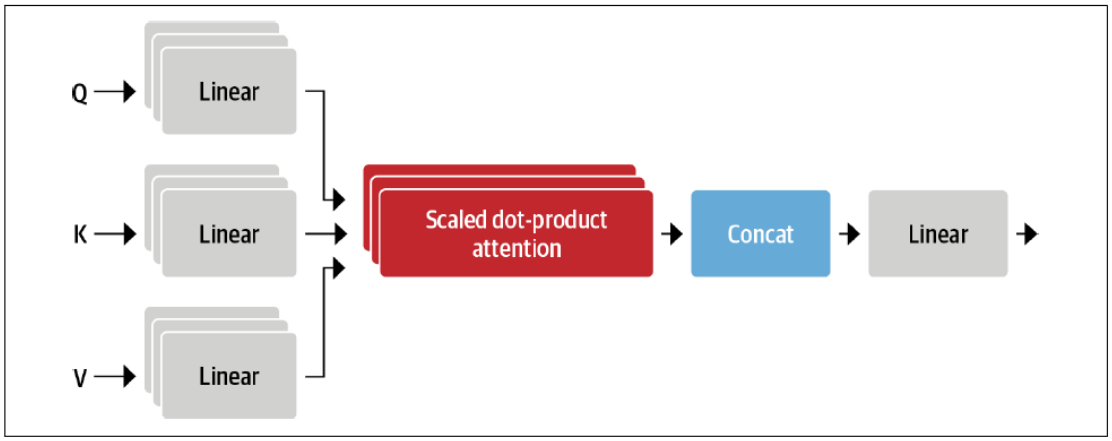


Figure 3-5. Multi-head attention

WHY??

The softmax of one head tends to focus on one aspect of similarity (it kills all other relations). Having several heads allows the model to focus on several aspects at once. For instance, one head can focus on subject-verb interaction, whereas another finds nearby adjectives.

This is similar to having multiple filters in CNNs for computer vision models where one filter can be responsible for detecting faces and another one finds wheels of cars in images.

[12]:
```
from torch import nn
import torch
import math



class MultiheadAttention(nn.Module):
    def __init__(self):
        super().__init__()
        self.inf = 1e9

        # W^Q, W^K, W^V in the paper
        self.w_q = nn.Linear(d_model, d_model)
        self.w_k = nn.Linear(d_model, d_model)
        self.w_v = nn.Linear(d_model, d_model)

        self.dropout = nn.Dropout(drop_out_rate)
        self.attn_softmax = nn.Softmax(dim=-1)

        # Final output linear transformation
        self.w_0 = nn.Linear(d_model, d_model)

    def forward(self, q, k, v, mask=None):
        input_shape = q.shape

        # Linear calculation +  split into num_heads
        q = self.w_q(q).view(input_shape[0], -1, num_heads, d_k) # (B, L,␣
↪num_heads, d_k)
        k = self.w_k(k).view(input_shape[0], -1, num_heads, d_k) # (B, L,␣
↪num_heads, d_k)
        v = self.w_v(v).view(input_shape[0], -1, num_heads, d_k) # (B, L,␣
↪num_heads, d_k)

        # For convenience, convert all tensors in size (B, num_heads, L, d_k)
        q = q.transpose(1, 2)
        k = k.transpose(1, 2)
        v = v.transpose(1, 2)

        # Conduct self-attention
```

```
        attn_values = self.self_attention(q, k, v, mask=mask) # (B, num_heads,␣
↪L, d_k)
        concat_output = attn_values.transpose(1, 2)\
            .contiguous().view(input_shape[0], -1, d_model) # (B, L, d_model)

        return self.w_O(concat_output)

    def self_attention(self, q, k, v, mask=None):
        # Calculate attention scores with scaled dot-product attention
        attn_scores = torch.matmul(q, k.transpose(-2, -1)) # (B, num_heads, L,␣
↪L)
        attn_scores = attn_scores / math.sqrt(d_k)

        # If there is a mask, make masked spots -INF
        if mask is not None:
            mask = mask.unsqueeze(1) # (B, 1, L) => (B, 1, 1, L) or (B, L, L)␣
↪=> (B, 1, L, L)
            attn_scores = attn_scores.masked_fill_(mask == 0, -1 * self.inf)

        # Softmax and multiplying K to calculate attention value
        attn_distribs = self.attn_softmax(attn_scores)

        attn_distribs = self.dropout(attn_distribs)
        attn_values = torch.matmul(attn_distribs, v) # (B, num_heads, L, d_k)

        return attn_values
```

4.0.2 The Feed-Forward Layer

The feed-forward sublayer in the encoder and decoder is just a simple two-layer fully connected neural network, but with a twist: instead of processing the whole sequence of embeddings as a single vector, it processes each embedding independently. For this reason, this layer is often referred to as a position-wise feed-forward layer.

[13]:
```
class FeedFowardLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_1 = nn.Linear(d_model, d_ff, bias=True)
        self.relu = nn.ReLU()
        self.linear_2 = nn.Linear(d_ff, d_model, bias=True)
        self.dropout = nn.Dropout(drop_out_rate)

    def forward(self, x):
        x = self.relu(self.linear_1(x)) # (B, L, d_ff)
        x = self.dropout(x)
        x = self.linear_2(x) # (B, L, d_model)
```

```
        return x
```

4.0.3 Adding Layer Normalization

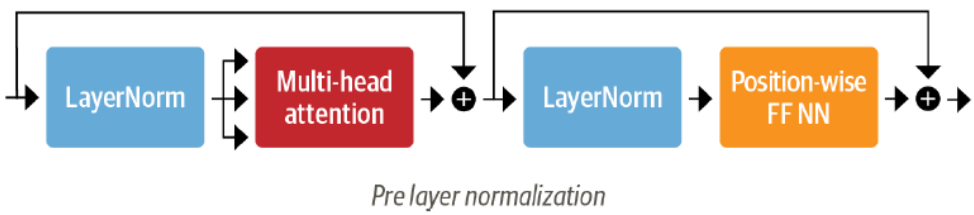
The layer normalization normalizes each input in the batch to have zero mean and unity variance.

$$y = \frac{x - E[x]}{\sqrt{V[x] + \epsilon}} \times \gamma + \beta$$

.

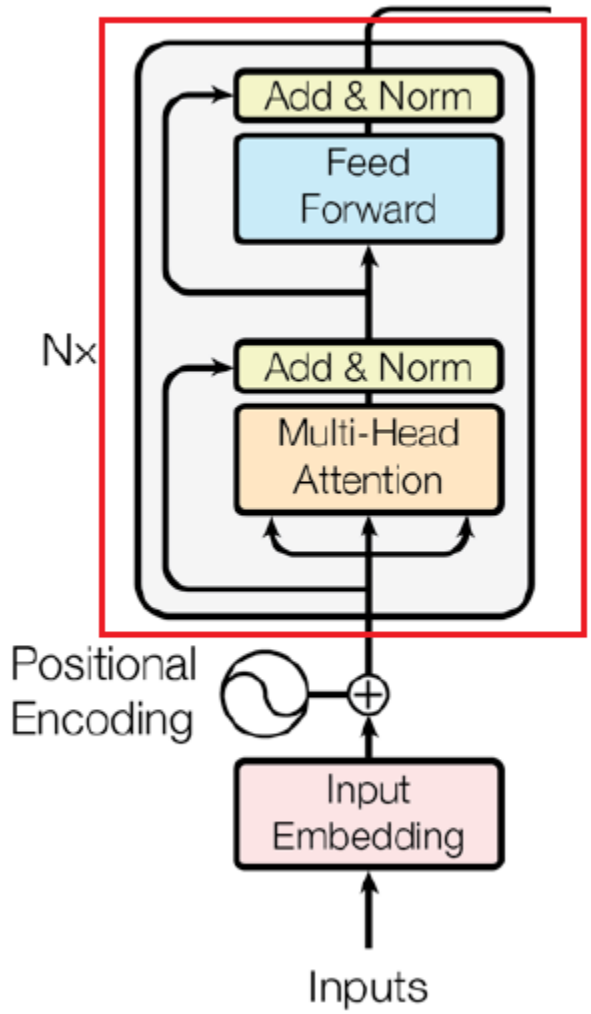
Skip connections pass a tensor to the next layer of the model without processing and add it to the processed tensor.

$y = x + Sublayer(x)$.



[14]:
```
class LayerNormalization(nn.Module):
    def __init__(self, eps=1e-6):
        super().__init__()
        self.eps = eps
        self.layer = nn.LayerNorm([d_model], elementwise_affine=True, eps=self.
 ↪eps)

    def forward(self, x):
        x = self.layer(x)

        return x
```

4.0.4 Complete Encoder Layer



[15]:
```
class EncoderLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer_norm_1 = LayerNormalization()
        self.multihead_attention = MultiheadAttention()
        self.drop_out_1 = nn.Dropout(drop_out_rate)

        self.layer_norm_2 = LayerNormalization()
        self.feed_forward = FeedFowardLayer()
        self.drop_out_2 = nn.Dropout(drop_out_rate)

    def forward(self, x, e_mask):
```

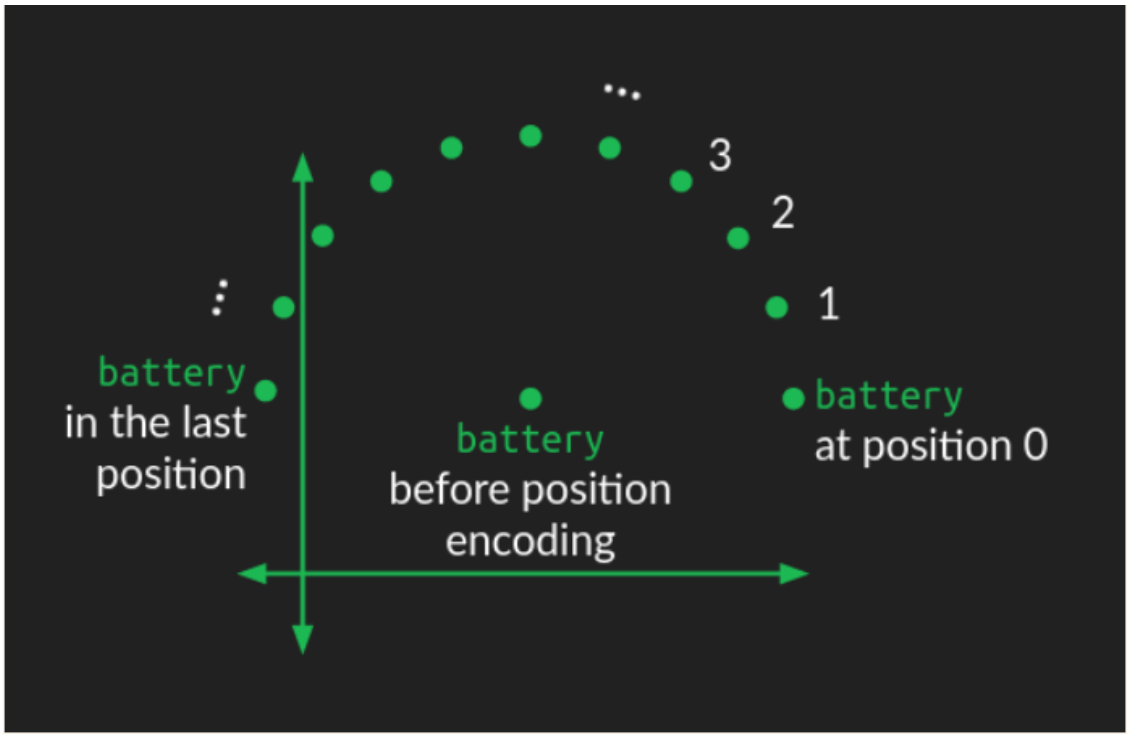
```
        x_1 = self.layer_norm_1(x) # (B, L, d_model)
        x = x + self.drop_out_1(
            self.multihead_attention(x_1, x_1, x_1, mask=e_mask)
        ) # (B, L, d_model)
        x_2 = self.layer_norm_2(x) # (B, L, d_model)
        x = x + self.drop_out_2(self.feed_forward(x_2)) # (B, L, d_model)

        return x # (B, L, d_model)
```

4.1 Complete Encoder

[16]:
```
class Encoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.ModuleList([EncoderLayer() for i in range(num_layers)])
        self.layer_norm = LayerNormalization()

    def forward(self, x, e_mask):
        for i in range(num_layers):
            x = self.layers[i](x, e_mask)

        return self.layer_norm(x)
```

4.1.1 Positional Embeddings

There are several ways that position information could be introduced into our embedded represetation of words, but the way it was done in the original transformer was to add a circular wiggle.



Positional Encoding Matrix with d=4, n=100

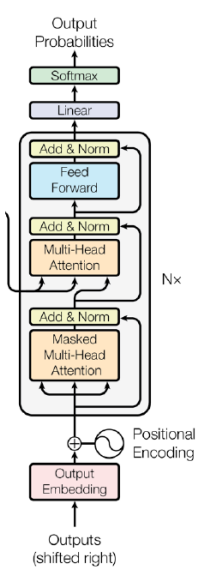
Sequence	Index of token, k	i=0	i=0	i=1	i=1
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

[17]:
```
class PositionalEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        # Make initial positional encoding matrix with 0
        pe_matrix= torch.zeros(seq_len, d_model) # (L, d_model)
```

```
        # Calculating position encoding values
        for pos in range(seq_len):
            for i in range(d_model):
                if i % 2 == 0:
                    pe_matrix[pos, i] = math.sin(pos / (10000 ** (2 * i /␣
↪d_model)))
                elif i % 2 == 1:
                    pe_matrix[pos, i] = math.cos(pos / (10000 ** (2 * i /␣
↪d_model)))

        pe_matrix = pe_matrix.unsqueeze(0) # (1, L, d_model)
        self.positional_encoding = pe_matrix.to(device=device).
↪requires_grad_(False)

    def forward(self, x):
        #x = x * math.sqrt(d_model) # (B, L, d_model)
        x = x + self.positional_encoding # (B, L, d_model)

        return x
```

4.2 The Decoder



Transformer decoder

Each decoder layer has three sub-layers:

- A **masked multi-head attention** layer
- A multi-head **cross-attention** layer
- A feedforward layer

Masked multi-head attention:
self-attention on the decoder states

However, you can't see the future!

Multi-head cross-attention:
Decoder attends to encoder states

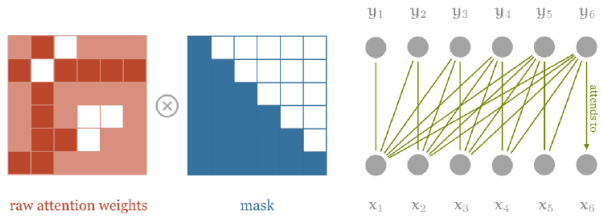
encoder: keys/values, decoder: queries

In (Vaswani et al., 2017), N = 6

Masked multi-head attention

$$\mathbf{q}_i = W^Q \mathbf{x}_i, \mathbf{k}_i = W^K \mathbf{x}_i, \mathbf{v}_i = W^V \mathbf{x}_i$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$



Efficient implementation: compute attention as we normally do, mask out attention to future words by setting attention scores to $-\infty$

Multi-head cross-attention

$$\mathbf{q}_i = W^Q \mathbf{x}_i, \mathbf{k}_i = W^K \mathbf{x}_i, \mathbf{v}_i = W^V \mathbf{x}_i$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$

- $\mathbf{h}_1, \ldots, \mathbf{h}_m$: hidden states from encoder
- $\mathbf{x}_1, \ldots, \mathbf{x}_n$: hidden states from decoder

$$\mathbf{q}_i = W^Q \mathbf{x}_i \quad \mathbf{k}_j = W^K \mathbf{h}_j, \mathbf{v}_j = W^V \mathbf{h}_j$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$

Q: What is the size of α?

$$\mathbf{y}_i = \sum_{j=1}^{m} \alpha_{i,j} \mathbf{v}_j$$

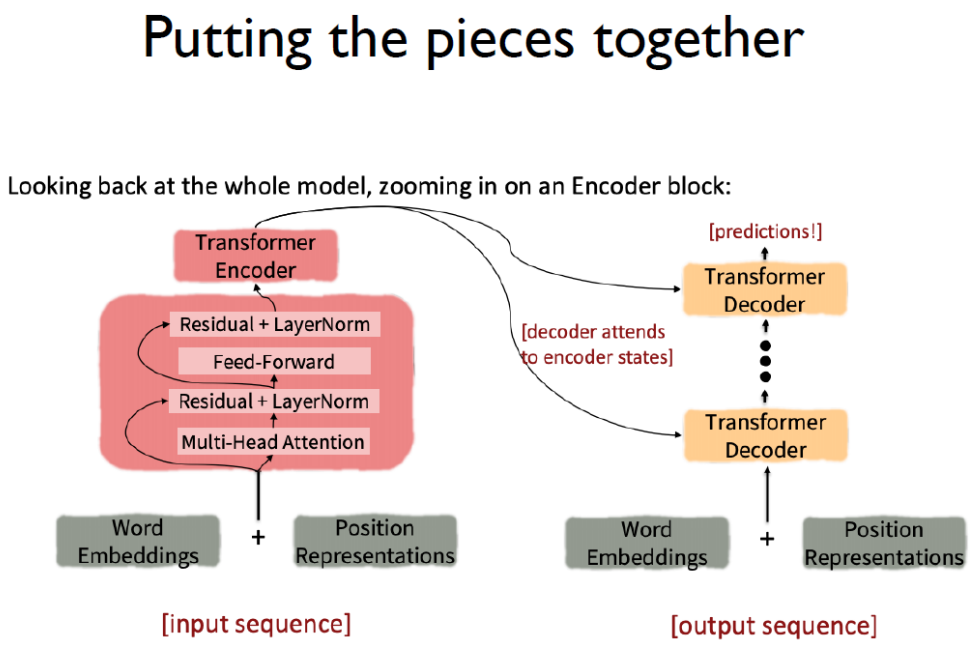
[18]:
```
class DecoderLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer_norm_1 = LayerNormalization()
        self.masked_multihead_attention = MultiheadAttention()
        self.drop_out_1 = nn.Dropout(drop_out_rate)

        self.layer_norm_2 = LayerNormalization()
```

```
        self.multihead_attention = MultiheadAttention()
        self.drop_out_2 = nn.Dropout(drop_out_rate)

        self.layer_norm_3 = LayerNormalization()
        self.feed_forward = FeedFowardLayer()
        self.drop_out_3 = nn.Dropout(drop_out_rate)

    def forward(self, x, e_output, e_mask,  d_mask):
        x_1 = self.layer_norm_1(x) # (B, L, d_model)
        x = x + self.drop_out_1(self.masked_multihead_attention(x_1, x_1, x_1,␣
↪mask=d_mask)) # (B, L, d_model)
        x_2 = self.layer_norm_2(x) # (B, L, d_model)
        x = x + self.drop_out_2(
            self.multihead_attention(x_2, e_output, e_output, mask=e_mask)
        ) # (B, L, d_model)
        x_3 = self.layer_norm_3(x) # (B, L, d_model)
        x = x + self.drop_out_3(self.feed_forward(x_3)) # (B, L, d_model)

        return x # (B, L, d_model)
```

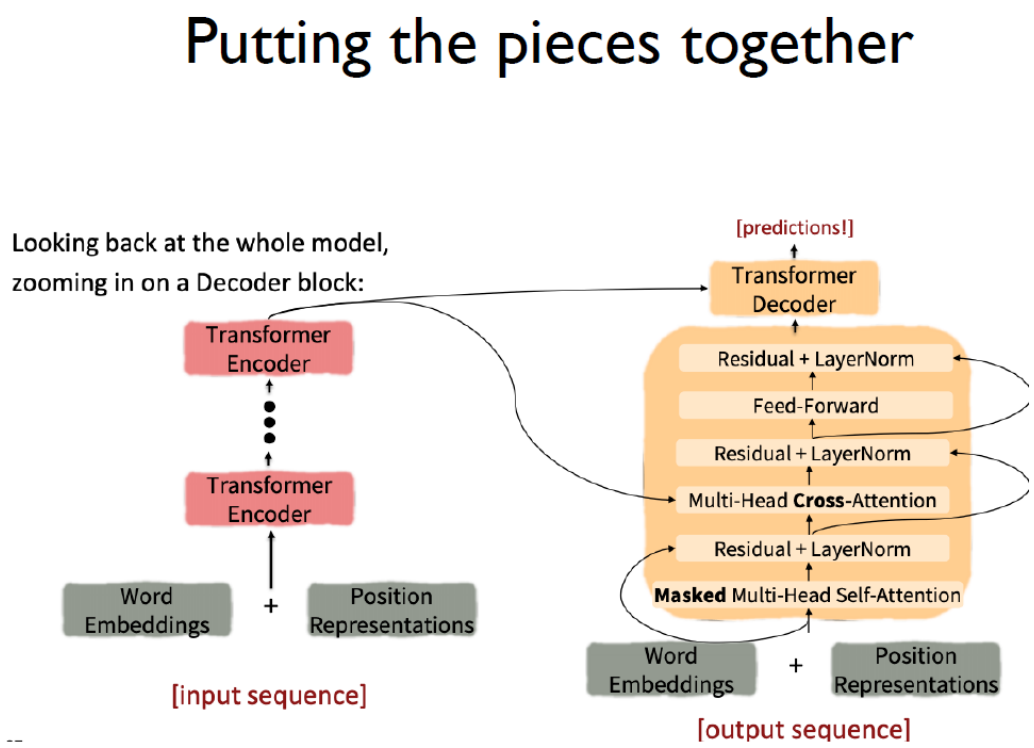
4.2.1 The complete Decoder

[19]:
```
class Decoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.ModuleList([DecoderLayer() for i in range(num_layers)])
        self.layer_norm = LayerNormalization()

    def forward(self, x, e_output, e_mask, d_mask):
        for i in range(num_layers):
            x = self.layers[i](x, e_output, e_mask, d_mask)

        return self.layer_norm(x)
```

4.2.2 Recap 1

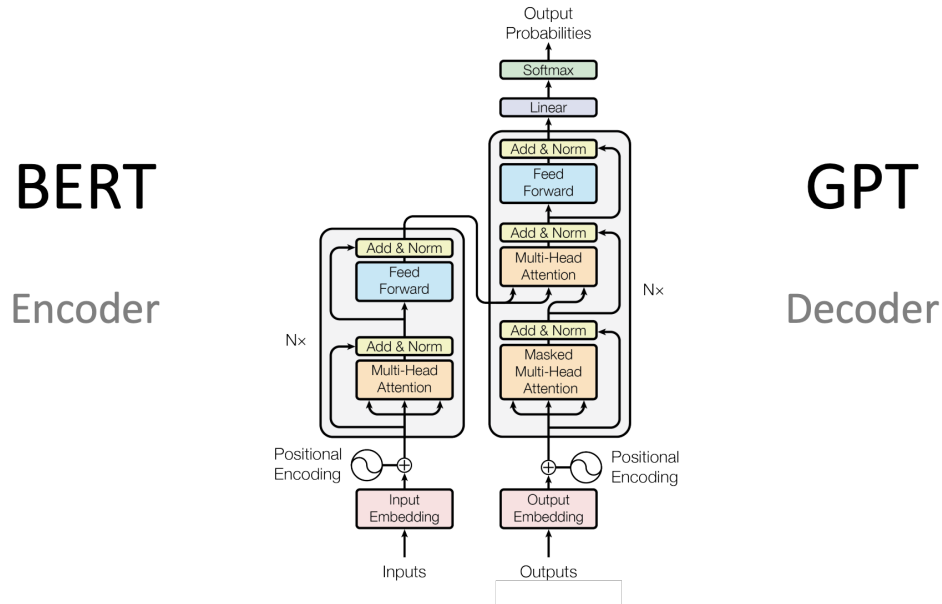


4.2.3 Recap 2



--

4.2.4 The full Transformer

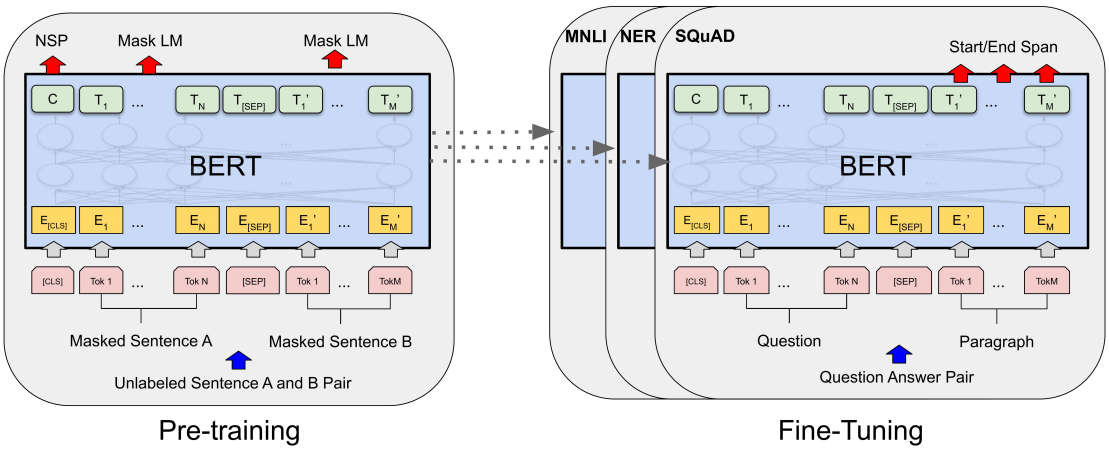


[20]:
```
class Transformer(nn.Module):
    def __init__(self, src_vocab_size, trg_vocab_size):
        super().__init__()
        self.src_vocab_size = src_vocab_size
        self.trg_vocab_size = trg_vocab_size

        self.src_embedding = nn.Embedding(self.src_vocab_size, d_model)
        self.trg_embedding = nn.Embedding(self.trg_vocab_size, d_model)
        self.positional_encoder = PositionalEncoder()
        self.encoder = Encoder()
        self.decoder = Decoder()
        self.output_linear = nn.Linear(d_model, self.trg_vocab_size)
        self.softmax = nn.LogSoftmax(dim=-1)

    def forward(self, src_input, trg_input, e_mask=None, d_mask=None):
        src_input = self.src_embedding(src_input) # (B, L) => (B, L, d_model)
        trg_input = self.trg_embedding(trg_input) # (B, L) => (B, L, d_model)
        src_input = self.positional_encoder(src_input) # (B, L, d_model) => (B,␣
↪L, d_model)
        trg_input = self.positional_encoder(trg_input) # (B, L, d_model) => (B,␣
↪L, d_model)

        e_output = self.encoder(src_input, e_mask) # (B, L, d_model)
        d_output = self.decoder(trg_input, e_output, e_mask, d_mask) # (B, L,␣
↪d_model)
```

```
        output = self.softmax(self.output_linear(d_output)) # (B, L, d_model)␣
↪=> # (B, L, trg_vocab_size)

        return output
```

4.3 Improvistions

4.3.1 Pretraining and Fine-tuning

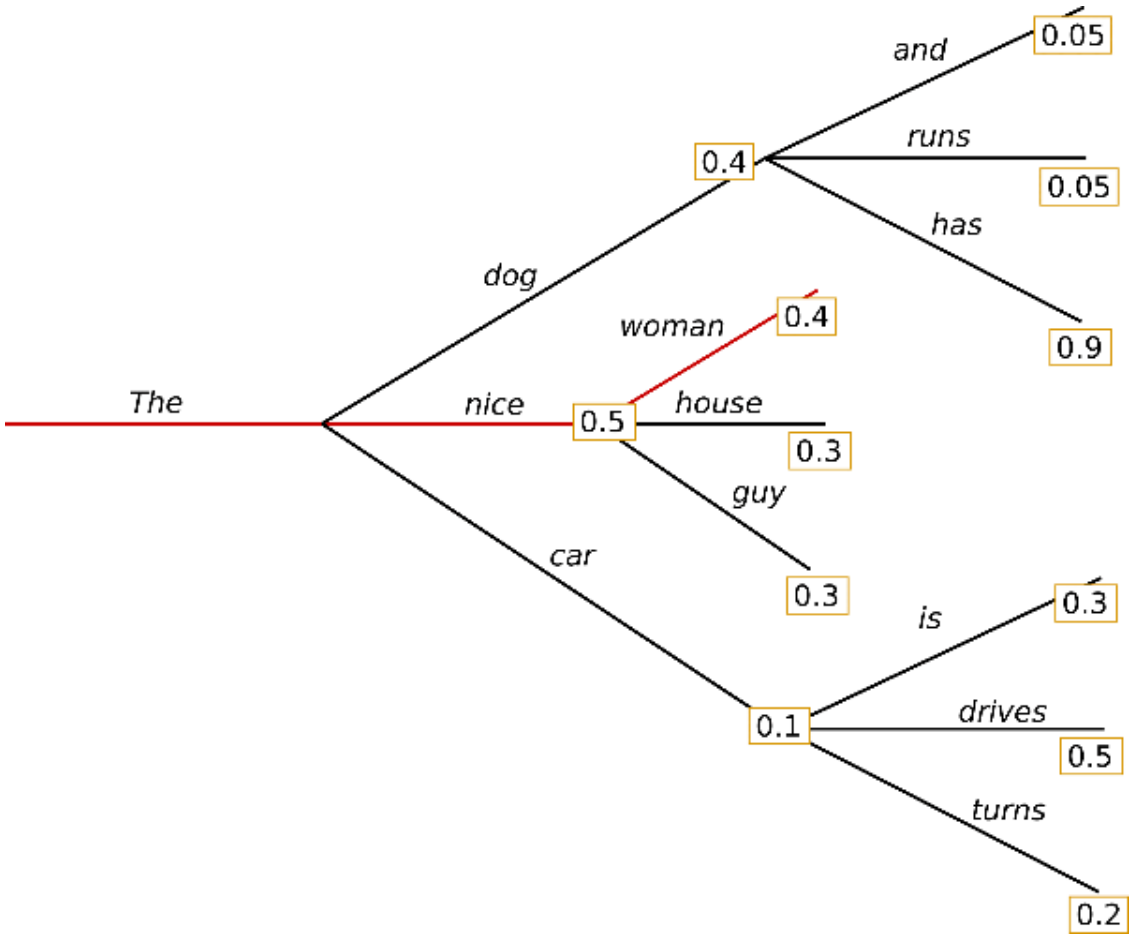


source

4.3.2 Decoding Schemes - 1

Code Open in Colab

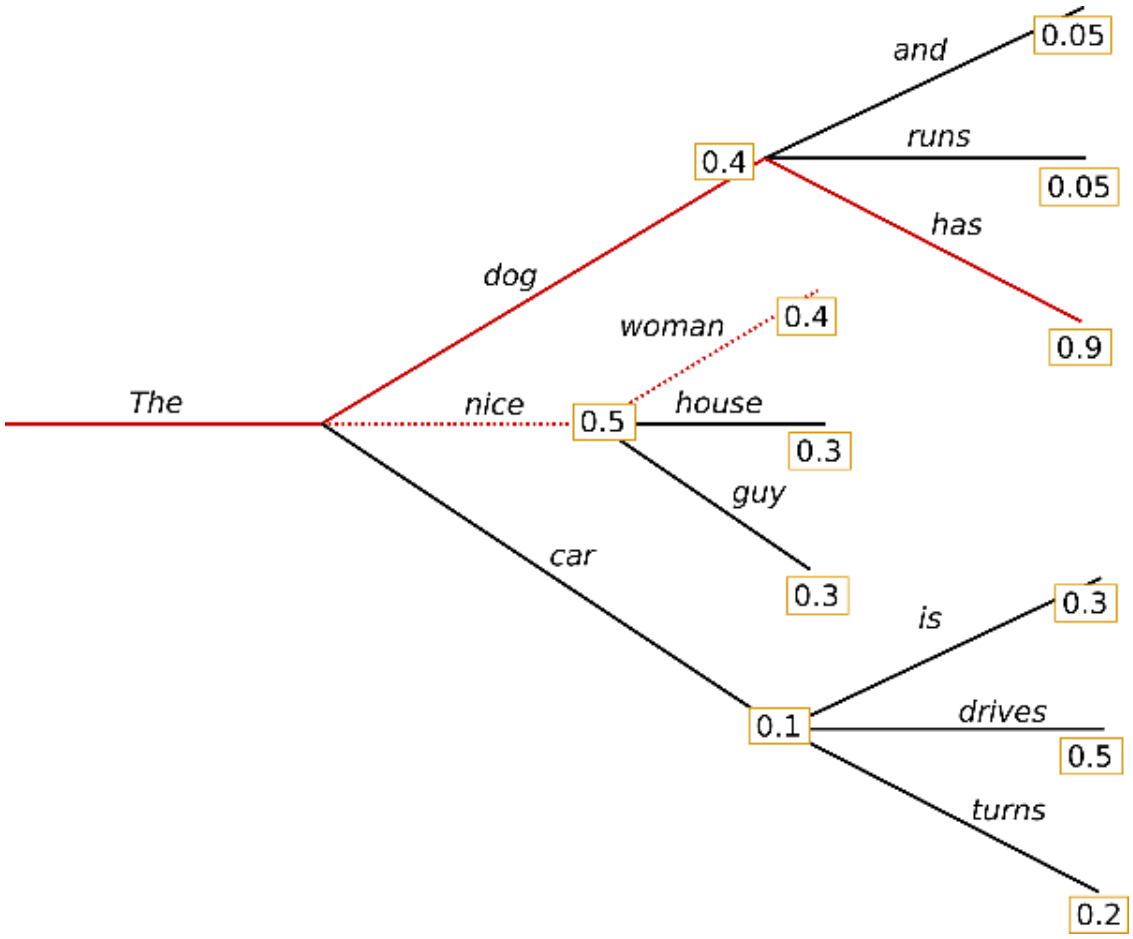
Greedy search simply selects the word with the highest probability as its next word: $w_t = argmax_w P(w|w_{1:t-1})$ at each timestep t.



Starting from the word "The", the algorithm greedily chooses the next word of highest probability "nice" and so on, so that the final generated word sequence is "The", "nice", "woman" having an overall probability of $0.5 \times 0.4 = 0.2$. In the following we will generate word sequences using GPT2 on the context ("I", "enjoy", "walking", "with", "my", "cute", "dog").

4.3.3 Decoding Schemes - 2

Beam search reduces the risk of missing hidden high probability word sequences by keeping the most likely `num_beams` of hypotheses at each time step and eventually choosing the hypothesis that has the overall highest probability. Let's illustrate with `num_beams=2`:



At time step 1, besides the most likely hypothesis "The", "nice", beam search also keeps track of the second most likely one "The", "dog". At time step 2, beam search finds that the word sequence "The", "dog", "has" has with 0.36 a higher probability than "The", "nice", "woman", which has 0.2. Great, it has found the most likely word sequence in our toy example!

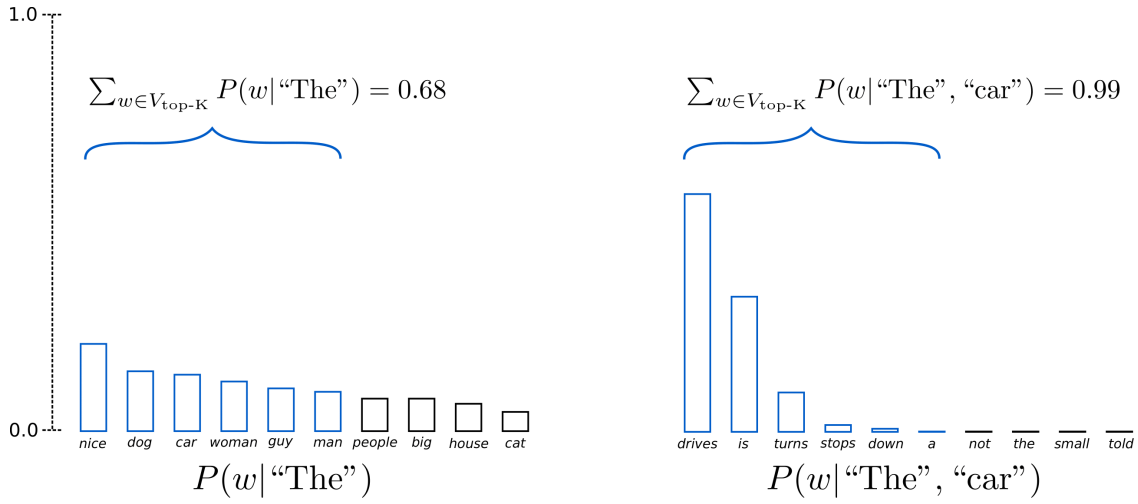
Beam search will always find an output sequence with higher probability than greedy search, but is not guaranteed to find the most likely output.

4.3.4 Decoding Schemes - 3

Top-K Sampling

Fan et. al (2018) introduced a simple, but very powerful sampling scheme, called ***Top-K*** sampling. In *Top-K* sampling, the K most likely next words are filtered and the probability mass is redistributed among only those K next words. GPT2 adopted this sampling scheme, which was one of the reasons for its success in story generation.

We extend the range of words used for both sampling steps in the example above from 3 words to 10 words to better illustrate *Top-K* sampling.

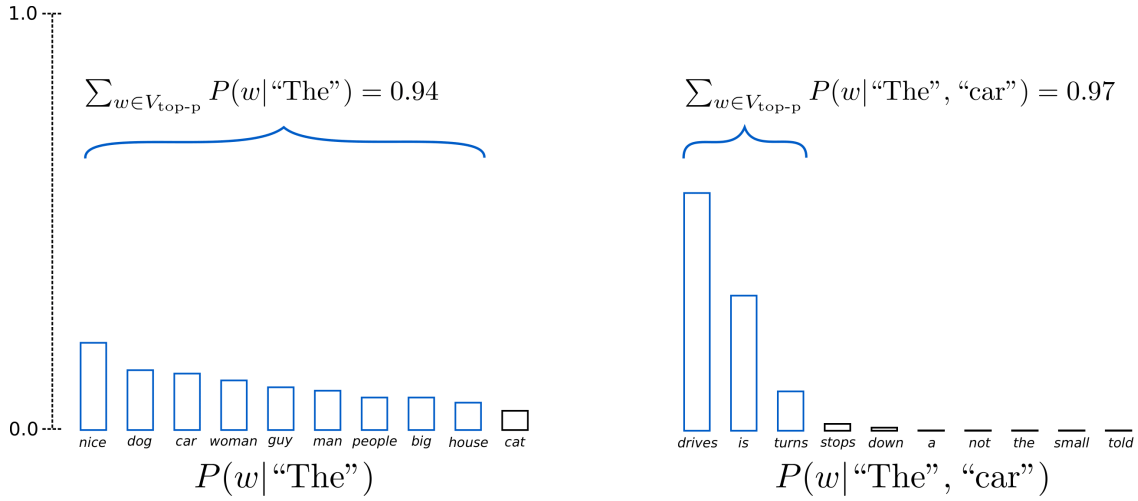


Having set $K = 6$, in both sampling steps we limit our sampling pool to 6 words. While the 6 most likely words, defined as $V_{\text{top-K}}$ encompass only *ca.* two-thirds of the whole probability mass in the first step, it includes almost all of the probability mass in the second step. Nevertheless, we see that it successfully eliminates the rather weird candidates ”not”, ”the”, ”small”, ”told” in the second sampling step.

4.3.5 Decoding Schemes - 4

Top-p (nucleus) sampling

Instead of sampling only from the most likely K words, in *Top-p* sampling chooses from the smallest possible set of words whose cumulative probability exceeds the probability p. The probability mass is then redistributed among this set of words. This way, the size of the set of words (*a.k.a* the number of words in the set) can dynamically increase and decrease according to the next word's probability distribution. Ok, that was very wordy, let's visualize.



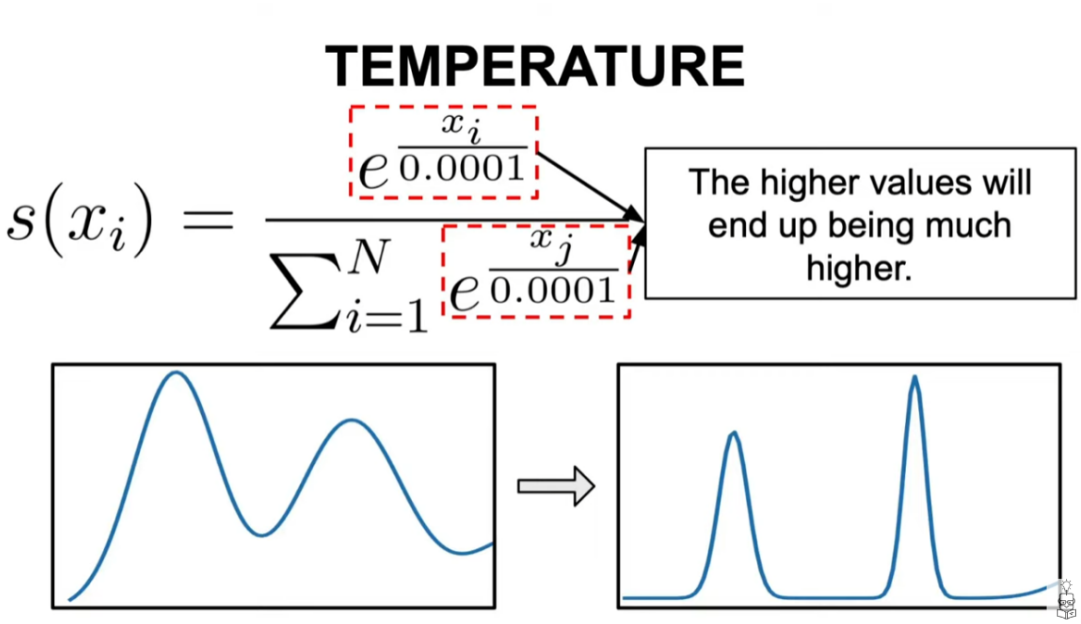
Having set $p = 0.92$, *Top-p* sampling picks the *minimum* number of words to exceed together $p = 92\%$ of the probability mass, defined as $V_{\text{top-p}}$. In the first example, this included the 9 most

likely words, whereas it only has to pick the top 3 words in the second example to exceed 92%. Quite simple actually! It can be seen that it keeps a wide range of words where the next word is arguably less predictable, *e.g.* $P(w|\text{"The"})$, and only a few words when the next word seems more predictable, *e.g.* $P(w|\text{"The", "car"})$.

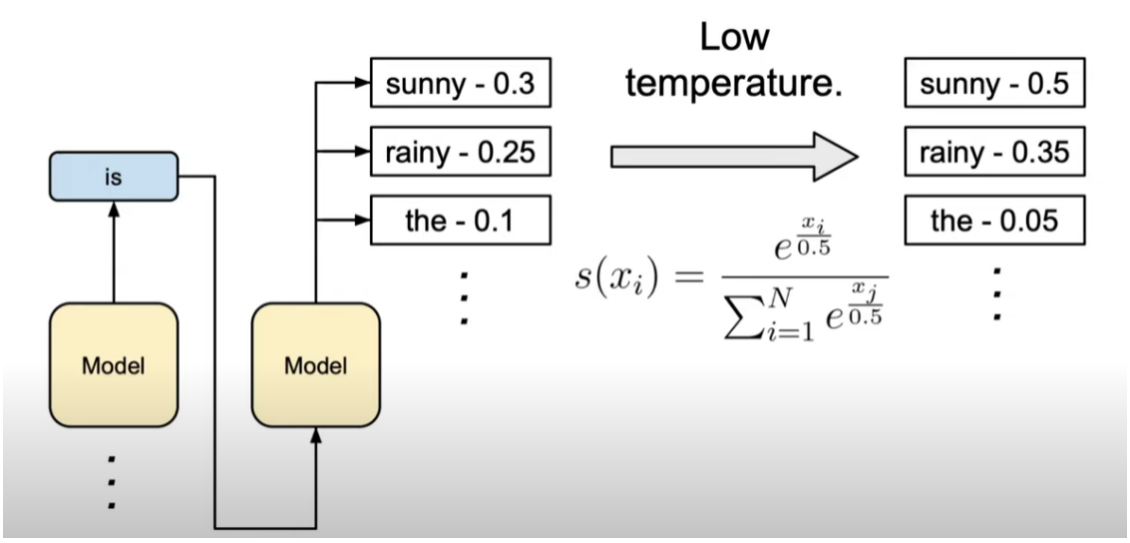
4.3.6 Decoding Parameter

Temperature This controls the randomness in the sampling process.

Softmax $s(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$



source



4.4 LLM

The loss scales as a power-law with model size, dataset size, and the amount of compute used for training. Increase model size 10 times, then increase data size by 2 times.

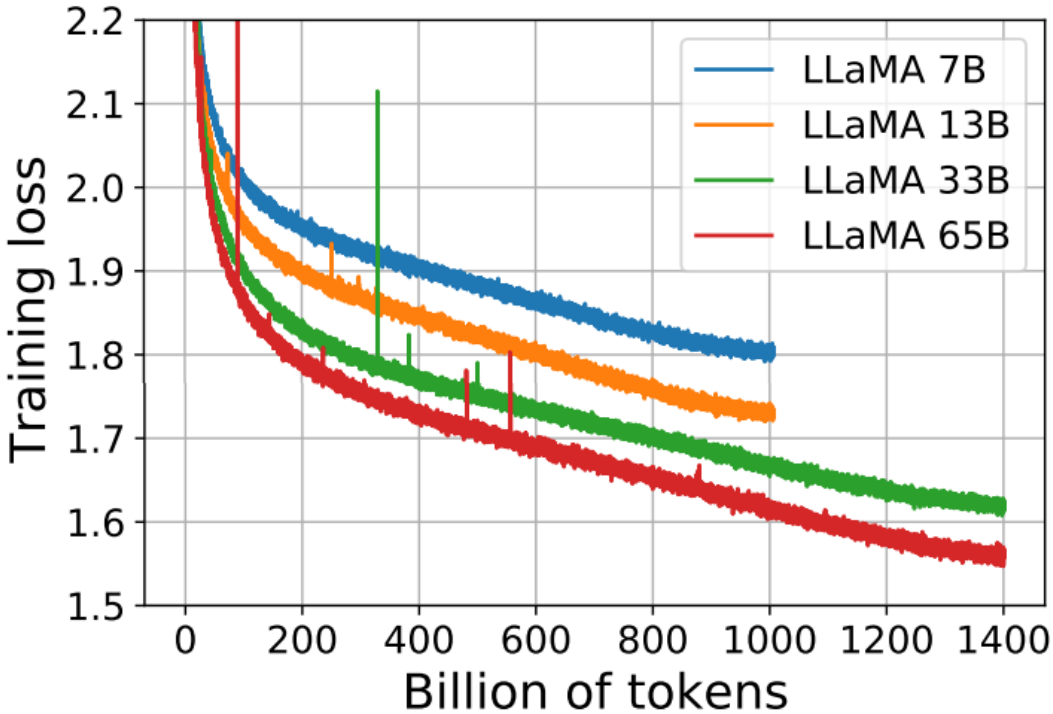
Kaplan, Jared, et al. "Scaling laws for neural language models." arXiv preprint arXiv:2001.08361 (2020).

Actually increase data by 10 times. Instead of bigger model, use more data and bigger batch size (compute).

Hoffmann, Jordan, et al. "Training compute-optimal large language models." arXiv preprint arXiv:2203.15556 (2022).

Careful data pruning can help. source.

4.5 LLM



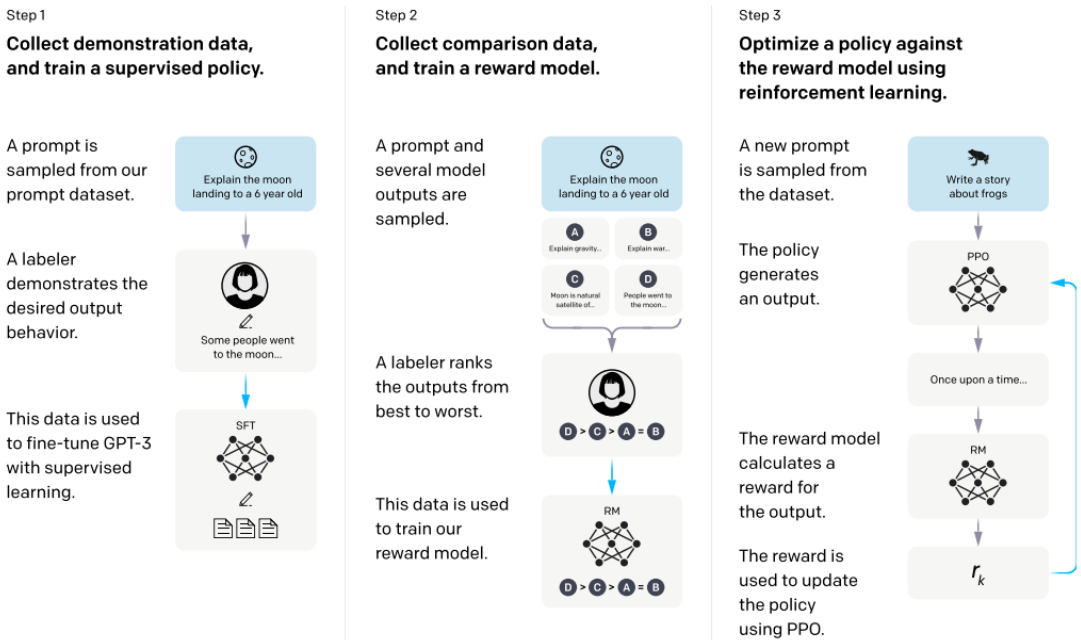
source

Engineering tricks used in Llama paper:

1. Pre-normalization
2. SwiGLU activation function
3. Rotary Embeddings (RoPE)
4. Gradinet clipping
5. Custom function for gradient backprop

65B parameter model needs 21 days to train on 2000+ A100 GPU with 1.4T tokens. Each GPU is 80 GB in size and 1.2 million INR in cost.

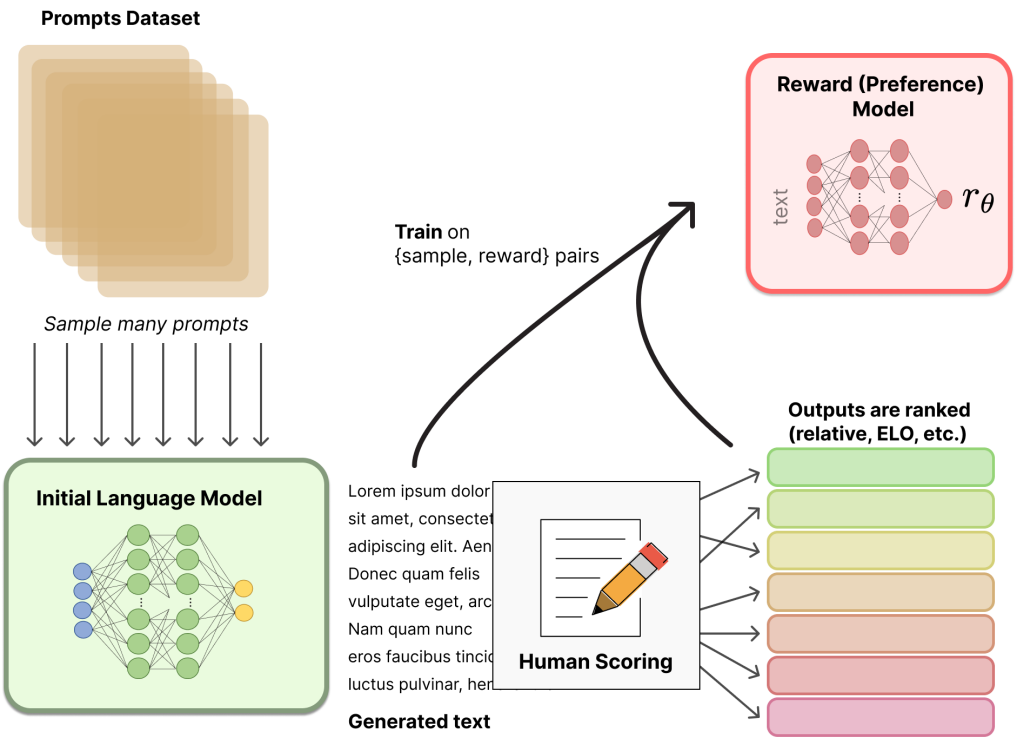
4.6 Instruction following LLMs



source

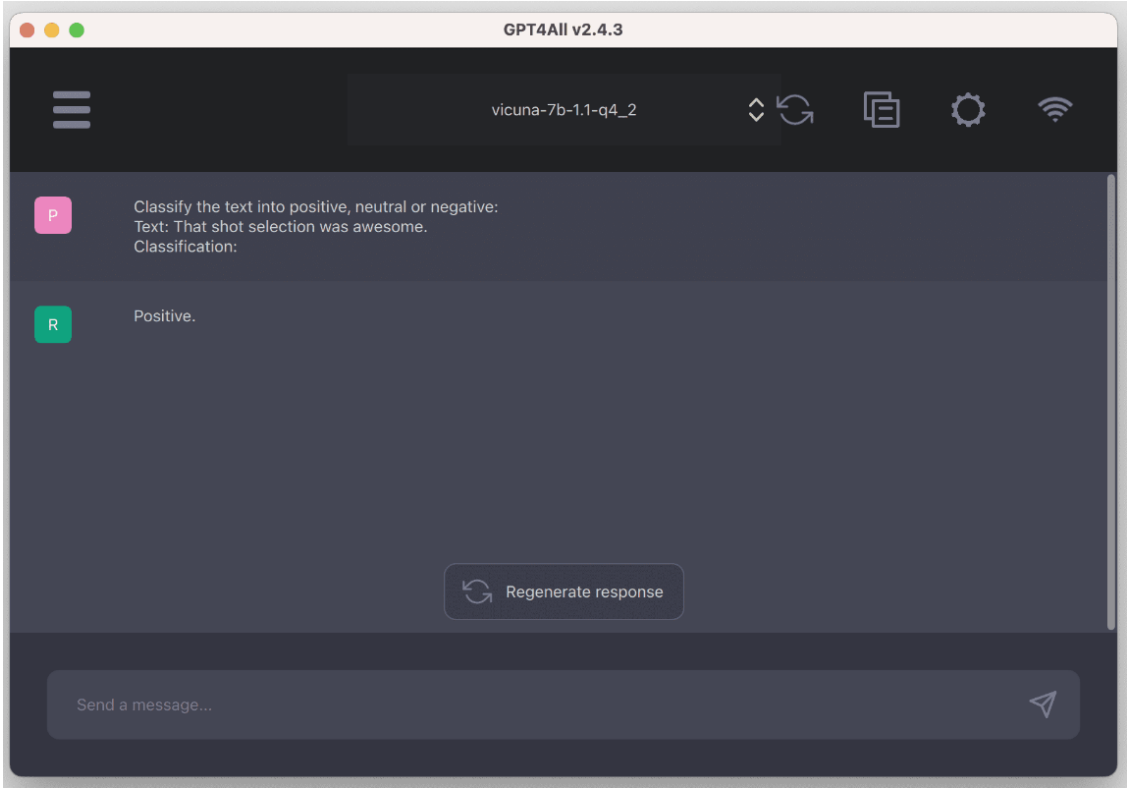
PPO resources: source1, source2

4.7 RLHF

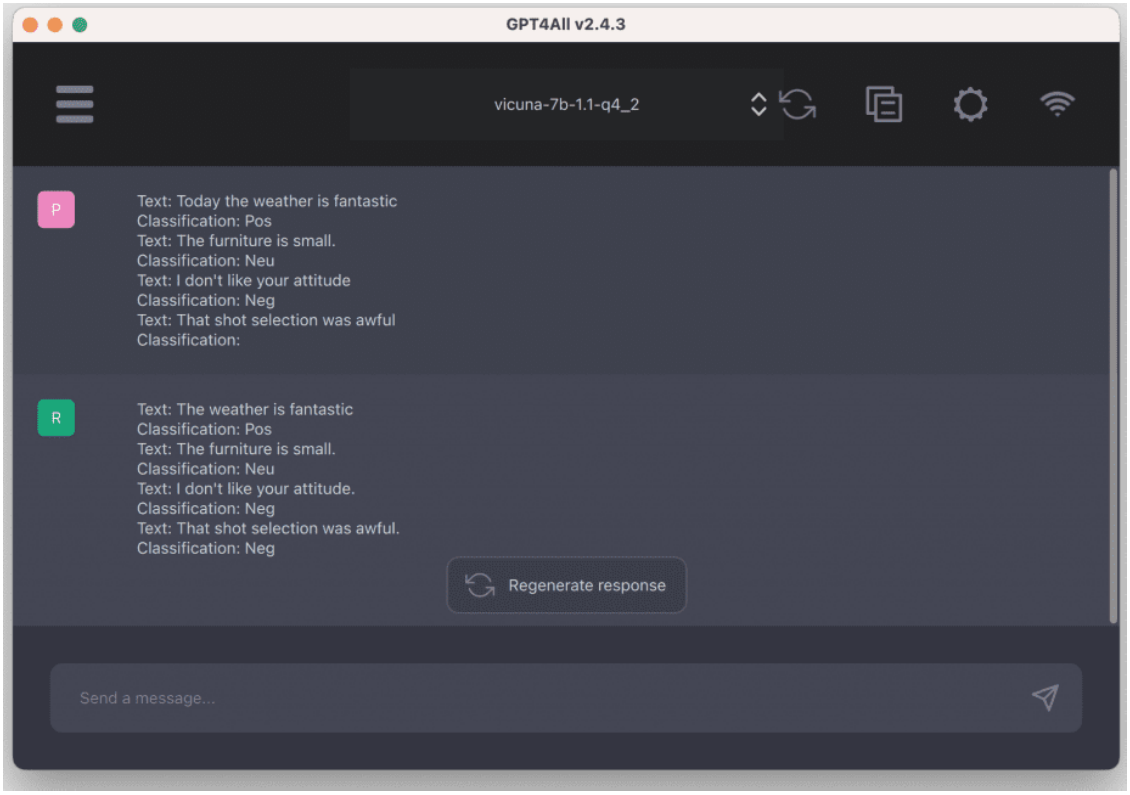


source

4.7.1 Zero-shot

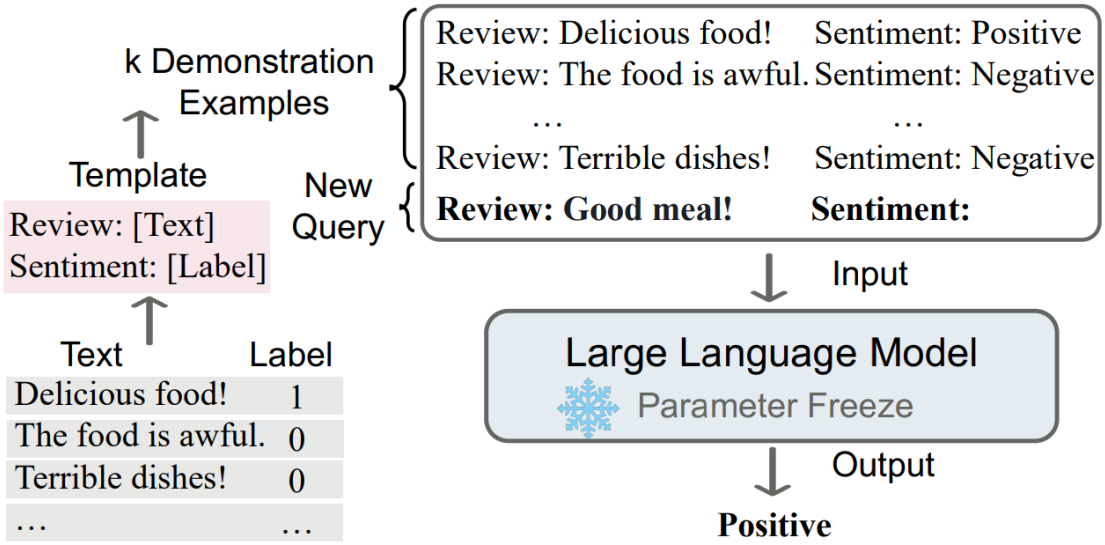


4.7.2 Few-shot



source

4.7.3 In-context



source

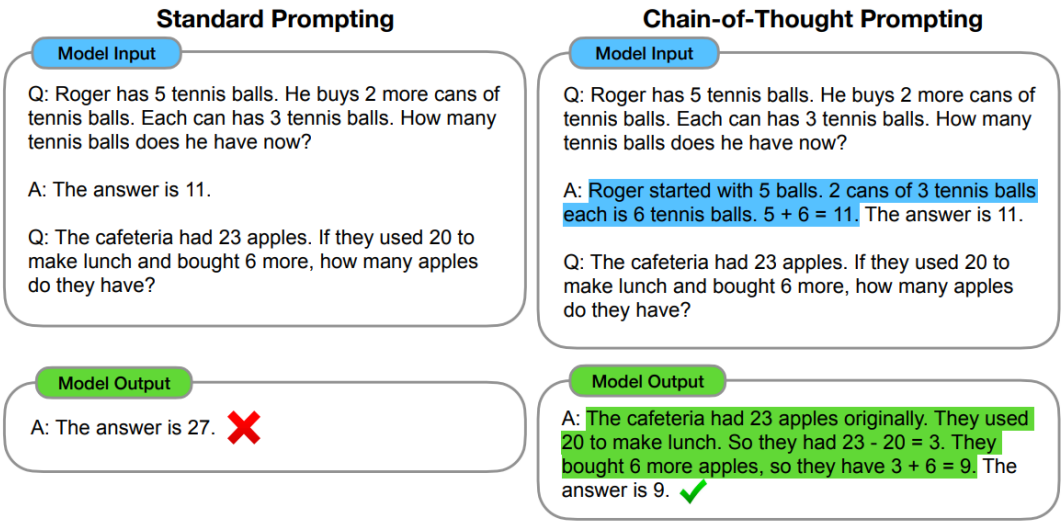
Incontext learning is a paradigm that allows language models to learn tasks given only a few examples in the form of demonstration. Few-shot learning is a general machine learning approach

that uses parameter adaptation to learn the best model parameters for the task with a limited number of supervised examples (Wang and Yao, 2019). In contrast, ICL does not require parameter updates and is directly performed on pretrained LLMs. source

Though it has been observed that performance does not vary much between few-shot and in-context.

Mosbach, Marius, et al. "Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation." arXiv preprint arXiv:2305.16938 (2023).

4.7.4 Chain of thought prompting



source

4.7.5 Prompting Notebooks

Generate positive reviews source

ILQL resource link

Detoxify Language Models link

4.8 Example of Simple Language Models

[1]:
```
# Import necessary libraries
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from nltk.tokenize import word_tokenize
from collections import Counter
import requests
```

```python
import nltk
from tqdm import tqdm
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[1]: True

[2]:
```python
# Download the dataset
#Linux
#!wget https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
 ↪tinyshakespeare/input.txt -O tinyshakespeare.txt
#Windows
import urllib.request
url = 'https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
 ↪tinyshakespeare/input.txt'
filename = 'tinyshakespeare.txt'
urllib.request.urlretrieve(url, filename)



# Load and preprocess the dataset
def load_data(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        text = f.read()
    return text

def build_vocab(text):
    tokens = word_tokenize(text)
    vocab = Counter(tokens)
    vocab = sorted(vocab, key=vocab.get, reverse=True)
    word_to_ix = {word: i for i, word in enumerate(vocab, 2)}
    word_to_ix['<PAD>'] = 0
    word_to_ix['<SOS>'] = 1
    ix_to_word = {i: word for word, i in word_to_ix.items()}
    return word_to_ix, ix_to_word, tokens

def encode_text(tokens, word_to_ix):
    return [word_to_ix[word] for word in tokens]

text = load_data('tinyshakespeare.txt')
word_to_ix, ix_to_word, tokens = build_vocab(text)
encoded_text = encode_text(tokens, word_to_ix)
```

[3]:
```
# Hyperparameters
vocab_size = len(word_to_ix)
embed_dim = 64
hidden_dim = 128
num_layers = 2
num_epochs = 5
batch_size = 64
learning_rate = 0.005

# Dataset and DataLoader
class ShakespeareDataset(Dataset):
    def __init__(self, encoded_text, seq_length):
        self.encoded_text = encoded_text
        self.seq_length = seq_length

    def __len__(self):
        return len(self.encoded_text) - self.seq_length

    def __getitem__(self, index):
        return (torch.tensor([word_to_ix['<SOS>']] + self.encoded_text[index:
 ↪index+self.seq_length-1]),
                torch.tensor(self.encoded_text[index:index+self.seq_length]))

dataset = ShakespeareDataset(encoded_text, seq_length=30)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

[4]:
```
class LinearModel(nn.Module):
    def __init__(self, vocab_size, embed_dim):
        super(LinearModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.linear = nn.Linear(embed_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        x = self.linear(x)
        return x
```

[5]:
```
class RNNModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers):
        super(RNNModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.rnn = nn.RNN(embed_dim, hidden_dim, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        x, _ = self.rnn(x)
```

```
        x = x[:, -1, :]
        x = self.linear(x)
        return x
```

[6]:
```
class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_layers):
        super(LSTMModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        x, (hn, cn) = self.lstm(x)
        x = x[:, -1, :]
        x = self.linear(x)
        return x
```

[7]:
```
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_heads, num_layers, dropout=0.
 ↪1):
        super(TransformerModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        encoder_layer = nn.TransformerEncoderLayer(d_model=embed_dim,␣
 ↪nhead=num_heads, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer,␣
 ↪num_layers=num_layers)
        self.pos_encoder = nn.Parameter(torch.zeros(1, 1000, embed_dim))
        self.linear = nn.Linear(embed_dim, vocab_size)

    def forward(self, x):
        seq_len, N = x.size()
        positions = torch.arange(0, seq_len).unsqueeze(1).expand(seq_len, N)
        x = self.embedding(x) + self.pos_encoder[:, :seq_len]
        x = self.transformer_encoder(x)
        x = x.mean(dim=0)
        x = self.linear(x)
        return x
```

[8]:
```
def train(model, dataloader, num_epochs, criterion, optimizer):
    model.train()
    for epoch in range(num_epochs):
        total_loss = 0
        for batch, (inputs, targets) in tqdm(enumerate(dataloader),␣
 ↪total=len(dataloader)):
            optimizer.zero_grad()
            outputs = model(inputs)
```

```
            outputs_reshaped = outputs.view(-1, outputs.shape[-1])
            targets_reshaped = targets.view(-1)
            loss = criterion(outputs_reshaped, targets_reshaped)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f'Epoch {epoch+1}, Loss: {total_loss/len(dataloader)}')
```

[9]:
```
def inference(model, start_sequence, word_to_ix, ix_to_word, sequence_length):
    """
    Performs inference with the language model.

    Parameters:
    - model: The trained language model.
    - start_sequence: The starting text sequence for generating new text.
    - word_to_ix: A dictionary mapping words to their integer indexes.
    - ix_to_word: A dictionary mapping integer indexes back to words.
    - sequence_length: The total length of the sequence to generate, including␣
↪the start sequence.

    Returns:
    - generated_text: The generated text sequence.
    """
    model.eval()  # Put the model in evaluation mode

    # Tokenize the start sequence
    tokens = start_sequence.split()
    generated_sequence = tokens.copy()

    # Convert tokens to their indexes
    current_sequence = [word_to_ix.get(word, word_to_ix['<PAD>']) for word in␣
↪tokens]

    # Generate words until we reach the desired sequence length
    while len(generated_sequence) < sequence_length:
        # Convert current sequence to tensor and add batch dimension
        sequence_tensor = torch.tensor([current_sequence], dtype=torch.long)

        # Get model output
        with torch.no_grad():
            output = model(sequence_tensor)

        # Get the last word distribution and convert to probabilities
        probabilities = torch.softmax(output[0, -1], dim=0).numpy()

        # Choose the word with the highest probability as the next word
        next_word_idx = np.argmax(probabilities)
```

```
        next_word = ix_to_word[next_word_idx]

        # Append the next word to the sequence
        generated_sequence.append(next_word)

        # Update the current sequence (considering the sequence length limit if␣
↪needed)
        current_sequence.append(next_word_idx)
        if len(current_sequence) > sequence_length:  # Optional: sliding window
            current_sequence = current_sequence[1:]

    generated_text = ' '.join(generated_sequence)
    return generated_text
```

[10]:
```
# Example for the Linear model
model1 = LinearModel(vocab_size, embed_dim)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model1.parameters(), lr=learning_rate)
train(model1, dataloader, num_epochs, criterion, optimizer)
```

```
100%|
  | 3977/3977 [18:06<00:00,  3.66it/s]

Epoch 1, Loss: 4.6794132092864125

100%|
  | 3977/3977 [17:57<00:00,  3.69it/s]

Epoch 2, Loss: 4.238905887253655

100%|
  | 3977/3977 [17:55<00:00,  3.70it/s]

Epoch 3, Loss: 4.192671326677314

100%|
  | 3977/3977 [17:56<00:00,  3.69it/s]

Epoch 4, Loss: 4.174423812231227

100%|
  | 3977/3977 [17:57<00:00,  3.69it/s]

Epoch 5, Loss: 4.1638349550346465
```

[11]:
```
inference(model1, "He that will give good words to ", word_to_ix, ix_to_word,␣
↪sequence_length=30)
```

[11]:
```
'He that will give good words to the king , and the king , and the king , and
the king , and the king , and the king ,'
```

[13]:
```
# Example for the Linear model
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
model2 = RNNModel(vocab_size, embed_dim, hidden_dim, num_layers)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model2.parameters(), lr=learning_rate)
train(model2, dataloader, num_epochs, criterion, optimizer)
```

```
  0%|
| 0/3977 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [13], in <cell line: 6>()
      4 criterion = nn.CrossEntropyLoss()
      5 optimizer = optim.Adam(model2.parameters(), lr=learning_rate)
----> 6 train(model2, dataloader, num_epochs, criterion, optimizer)

Input In [8], in train(model, dataloader, num_epochs, criterion, optimizer)
      8 outputs_reshaped = outputs.view(-1, outputs.shape[-1])
      9 targets_reshaped = targets.view(-1)
---> 10 loss = criterion(outputs_reshaped, targets_reshaped)
     11 loss.backward()
     12 optimizer.step()

File ~\miniconda3\envs\ML_\lib\site-packages\torch\nn\modules\module.py:1511, i␣
↪Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type:␣
↪ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~\miniconda3\envs\ML_\lib\site-packages\torch\nn\modules\module.py:1520, i␣
↪Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self.
↪_forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~\miniconda3\envs\ML_\lib\site-packages\torch\nn\modules\loss.py:1179, in␣
↪CrossEntropyLoss.forward(self, input, target)
   1178 def forward(self, input: Tensor, target: Tensor) -> Tensor:
-> 1179     return F.cross_entropy(input, target, weight=self.weight,
```

```
  1180␣
↪                                ignore_index=self.ignore_index, reduction=self.reduction,
  1181                                 label_smoothing=self.label_smoothing)

File ~\miniconda3\envs\ML_\lib\site-packages\torch\nn\functional.py:3059, in␣
↪cross_entropy(input, target, weight, size_average, ignore_index, reduce,␣
↪reduction, label_smoothing)
  3057 if size_average is not None or reduce is not None:
  3058     reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 3059 return␣
↪torch._C._nn.cross_entropy_loss(input, target, weight, _Reduction.get_enum(reduction), ign

ValueError: Expected input batch_size (64) to match target batch_size (1920).
```

[ ]:
```
inference(model2, "He that will give good words to ", word_to_ix, ix_to_word,␣
↪sequence_length=30)
```

[ ]:
```
model3= LSTMModel(vocab_size, embed_dim, hidden_dim, num_layers)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model3.parameters(), lr=learning_rate)
train(model3, dataloader, num_epochs, criterion, optimizer)
```

[ ]:
```
inference(model3, "He that will give good words to ", word_to_ix, ix_to_word,␣
↪sequence_length=30)
```

[ ]:
```
model4=TransformerModel(vocab_size, embed_dim, hidden_dim, num_layers)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model4.parameters(), lr=learning_rate)
train(model4, dataloader, num_epochs, criterion, optimizer)
```

[ ]:
```
inference(model4, "He that will give good words to ", word_to_ix, ix_to_word,␣
↪sequence_length=30)
```