Large Language Models
Recap: Attention and Transformers
SpeakUp

Join by URL: https://web.speakup.info/room/join/80740

Join by key: web.speakup.info → use key 80740

Join by saved session: web.speakup.info

Join by QR:
Outline

• Large Language Model (LLM) implementations

• Text continuation with GPT-2

• Fine-tuning LLMs
Transformers and Transfer Learning

In principle, the original GPT paper was only about the benefits of pre-training a transformer model for transfer learning. The paper showed that pre-training a 117M GPT achieved state-of-the-art performance on various NLP (natural language processing) tasks when fine-tuned on labelled datasets.

It wasn't until the GPT-2 and GPT-3 papers that we realized a GPT model pre-trained on enough data with enough parameters was capable of performing any arbitrary task by itself, no fine-tuning needed. Just prompt the model, perform autoregressive language modeling, and like voila, the model magically gives us an appropriate response. This is referred to as in-context learning, because the model is using just the context of the prompt to perform the task. In-context learning can be zero shot, one shot, or few shot:

Jay Mody
Some Examples

• Google: LaMDA: our breakthrough conversation technology
• OpenAI: Language Models are Few-Shot Learners (GPT-3)
  • nomic-ai/gpt4all: gpt4all: an ecosystem of open-source chatbots trained on a massive collections of clean assistant data including code, stories and dialogue
  • Cerebras-GPT: A Family of Open, Compute-efficient, Large Language Models - Cerebras
• BigScience (Open): A 176B-Parameter Open-Access Multilingual Language Model (BLOOM)
• DeepMind: A Generalist Agent (Gato)
• Together (Open): Announcing OpenChatKit
• Meta AI: Introducing LLaMA: A foundational, 65-billion-parameter language model
  • Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality | by the Team with members from UC Berkeley, CMU, Stanford, and UC San Diego
• Databricks: Free Dolly: Introducing the World's First Open and Commercially Viable Instruction-Tuned LLM - The Databricks Blog
• [2303.06865] High-throughput Generative Inference of Large Language Models with a Single GPU
## Model Scale

<table>
<thead>
<tr>
<th>Model</th>
<th>Access</th>
<th>Owner</th>
<th>Year</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>Open</td>
<td>Google</td>
<td>2016</td>
<td>213.0M</td>
</tr>
<tr>
<td>BERT</td>
<td>Open</td>
<td>Google</td>
<td>2018</td>
<td>340.0M</td>
</tr>
<tr>
<td>LaMDA</td>
<td>Closed</td>
<td>Google</td>
<td>2021</td>
<td>173.0B</td>
</tr>
<tr>
<td>FLAN UL2</td>
<td>Open</td>
<td>Google</td>
<td>2022</td>
<td>20.0B</td>
</tr>
<tr>
<td>U-PaLM</td>
<td>Semi-Open</td>
<td>Google</td>
<td>2022</td>
<td>540.0B</td>
</tr>
<tr>
<td>GPT-2</td>
<td>Open</td>
<td>OpenAI</td>
<td>2019</td>
<td>1.5B</td>
</tr>
<tr>
<td>GPT-3</td>
<td>Closed</td>
<td>OpenAI</td>
<td>2020</td>
<td>175.0B</td>
</tr>
<tr>
<td>ChatGPT</td>
<td>Closed</td>
<td>OpenAI</td>
<td>2022</td>
<td>20.0B</td>
</tr>
<tr>
<td>GPT-4</td>
<td>Closed</td>
<td>OpenAI</td>
<td>2023</td>
<td>1.0T</td>
</tr>
<tr>
<td>LLaMA</td>
<td>Semi-Open</td>
<td>Meta</td>
<td>2023</td>
<td>65.0B (multiple)</td>
</tr>
<tr>
<td>MT-NLG</td>
<td>Semi-Open</td>
<td>MS+NVIDIA</td>
<td>2021</td>
<td>530.0B</td>
</tr>
<tr>
<td>Claude</td>
<td>Semi-Open</td>
<td>Anthropic</td>
<td>2023</td>
<td>unknown</td>
</tr>
<tr>
<td>GATO</td>
<td>Closed</td>
<td>DeepMind</td>
<td>2022</td>
<td>1.2B</td>
</tr>
<tr>
<td>BLOOM</td>
<td>Open</td>
<td>BigScience</td>
<td>2022</td>
<td>176.0B</td>
</tr>
<tr>
<td>Alpaca</td>
<td>Open</td>
<td>Stanford</td>
<td>2023</td>
<td>7.0B</td>
</tr>
<tr>
<td>ChatGLM</td>
<td>Open</td>
<td>Tsinghua University</td>
<td>2023</td>
<td>6.0B</td>
</tr>
</tbody>
</table>

So many others: Baidu, Huawei, Cerebras, Bloomberg, ...

*Inside language models (from GPT-4 to PaLM) – Dr Alan D. Thompson – Life Architect*
<table>
<thead>
<tr>
<th>Model Name</th>
<th>Access Type</th>
<th>Owner</th>
<th>Year</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer</td>
<td>Open</td>
<td>Google</td>
<td>2016</td>
<td>213.0M</td>
</tr>
<tr>
<td>BERT</td>
<td>Open</td>
<td>Google</td>
<td>2018</td>
<td>340.0M</td>
</tr>
<tr>
<td>LaMDA</td>
<td>Closed</td>
<td>Google</td>
<td>2021</td>
<td>173.0B</td>
</tr>
<tr>
<td>FLAN UL2</td>
<td>Open</td>
<td>Google</td>
<td>2022</td>
<td>20.0B</td>
</tr>
<tr>
<td>U-PaLM</td>
<td>Semi-Open</td>
<td>Google</td>
<td>2022</td>
<td>540.0B</td>
</tr>
<tr>
<td>GPT-2</td>
<td>Open</td>
<td>OpenAI</td>
<td>2019</td>
<td>1.5B</td>
</tr>
<tr>
<td>GPT-3</td>
<td>Closed</td>
<td>OpenAI</td>
<td></td>
<td>175.0B</td>
</tr>
<tr>
<td>ChatGPT</td>
<td>Closed</td>
<td>OpenAI</td>
<td></td>
<td>20.0B</td>
</tr>
<tr>
<td>GPT-4</td>
<td>Closed</td>
<td>OpenAI</td>
<td></td>
<td>1.0T</td>
</tr>
<tr>
<td>LLaMA</td>
<td>Semi-Open</td>
<td>Meta</td>
<td>2023</td>
<td>65.0B</td>
</tr>
<tr>
<td>MT-NLG</td>
<td>Semi-Open</td>
<td>MS+NVIDIA</td>
<td>2021</td>
<td>530.0B</td>
</tr>
<tr>
<td>Claude</td>
<td>Semi-Open</td>
<td>Anthropic</td>
<td>2023</td>
<td>unknown</td>
</tr>
<tr>
<td>GATO</td>
<td>Closed</td>
<td>DeepMind</td>
<td>2022</td>
<td>1.2B</td>
</tr>
<tr>
<td>BLOOM</td>
<td>Open</td>
<td>BigScience</td>
<td>2022</td>
<td>176.0B</td>
</tr>
<tr>
<td>Alpaca</td>
<td>Open</td>
<td>Stanford</td>
<td>2023</td>
<td>7.0B</td>
</tr>
<tr>
<td>ChatGLM</td>
<td>Open</td>
<td>Tsinghua University</td>
<td>2023</td>
<td>6.0B</td>
</tr>
</tbody>
</table>

Many of these models need a GPU with ≥ 40 GB of video memory just to run.

Inside language models (from GPT-4 to PaLM) – Dr Alan D. Thompson – Life Architect
def main(prompt: str, length: int,
model_size: str = "124M",
model_dir: Path = Path("models"),
) -> tuple[str, str]:

# Load from OpenAI gpt-2 files
encoder, hparams, params = load_encoder_hparams_and_params(model_size, model_dir)

# Encode the input string
input_ids = encoder.encode(prompt)
assert len(input_ids) + length <= hparams["n_ctx"]

# Generate the output indices and decode into a string
output_ids = generate(input_ids, params, hparams["n_head"], length)
return prompt, encoder.decode(output_ids)
```python
def __download_gpt2_files(model_size: str, model_dir: Path) -> None:
    "Used by load_encoder_hparams_and_params."
files_to_download = [
    "checkpoint",
    "encoder.json",
    "hparams.json",
    "model.ckpt.data-00000-of-00001",
    "model.ckpt.index",
    "model.ckpt.meta",
    "vocab.bpe",
]

for filename in files_to_download:
    url = "https://openai-public.blob.core.windows.net/gpt-2/models"
    req = requests.get(f"{url}/{model_size}/{filename}", stream=True)

...
```python
def __download_gpt2_files(model_size: str, model_dir: Path) -> None:
    "Used by load_encoder_hparams_and_params."
    files_to_download = [
        "checkpoint",
        "encoder.json",
        "hparams.json",
        "model.ckpt.data-00000-of-00001",
        "model.ckpt.index",
        "model.ckpt.meta",
        "vocab.bpe",
    ]
```

```
l1 -h models/124M/
total 477M
-rw-r--r--. 1 ajcd2020 domain users  77 Apr 24 11:17 checkpoint
-rw-r--r--. 1 ajcd2020 domain users 1018K Apr 24 11:17 encoder.json
-rw-r--r--. 1 ajcd2020 domain users  90 Apr 24 11:17 hparams.json
-rw-r--r--. 1 ajcd2020 domain users 475M Apr 24 11:19 model.ckpt.data-00000-of-00001
-rw-r--r--. 1 ajcd2020 domain users  5.1K Apr 24 11:19 model.ckpt.index
-rw-r--r--. 1 ajcd2020 domain users 461K Apr 24 11:19 model.ckpt.meta
-rw-r--r--. 1 ajcd2020 domain users 446K Apr 24 11:19 vocab.bpe
```
def __download_gpt2_files(model_size: str, model_dir: Path) -> None:
    "Used by load_encoder_hparams_and_params."
    files_to_download = [
        "checkpoint",
        "encoder.json",
        "hparams.json",
        "model.ckpt.data-00000-of-00001",
        "model.ckpt.index",
    ]
    for filename in files_to_download:
        url = "https://openaipublic.blob.core.windows.net/gpt-2/models"
        url = url + "/{model_size}/{filename}".format(model_size=model_size, filename=filename)
        req = requests.get(url, stream=True)

        # more code...

    hparams = json.loads(open(hparams.json).read())
    n_vocab = 50257, # number of tokens in our vocabulary
    n_ctx = 1024, # maximum possible sequence length of the input
    n_embd = 768, # embedding dimension (determines the "width" of the network)
    n_head = 12, # number of attention heads (n_embd must be divisible by n_head)
    n_layer = 12 # number of layers (determines the "depth" of the network)

    wpe = [1024, 768],
    wte = [50257, 768],
    ln_f = {"b": [768], "g": [768]},
    blocks = [
        {
            "attn": {
                "c_attn": {"b": [2304], "w": [768, 2304]},
                "c_proj": {"b": [768], "w": [768, 768]},
            },
            "ln_1": {"b": [768], "g": [768]},
            "ln_2": {"b": [768], "g": [768]},
            "mlp": {
                "c_fc": {"b": [3072], "w": [768, 3072]},
                "c_proj": {"b": [768], "w": [3072, 768]},
            },
        },
        ... # repeat for n_layers
    ]
def generate(input_ids: Tokens, params: dict, n_head: int, output_length) -> Tokens:
    "Generate tokens from input tokens."
    # Auto-regressive decode loop
    for _ in tqdm(range(output_length), "generating"):
        logits = gpt2(input_ids, **params, n_head=n_head)
        next_id = np.argmax(logits[-1]) # greedy sampling
        input_ids.append(int(next_id))

    # Only return the generated tokens (not the input tokens)
    return input_ids[len(input_ids) - output_length :]
def gpt2(input_ids: Tokens, wte, wpe, blocks, ln_f, n_head: int) -> list[Float]:
    "Generate logits for next token."
    # Produce positional embeddings from input tokens [n_seq] -> [n_seq, n_embd]
    x = wte[input_ids] + wpe[range(len(input_ids))]

    for block in blocks:
        x = transformer_block(x, **block, n_head=n_head)
    x = layer_norm(x, **ln_f)

    # Project from embeddings to vocabulary logits [n_seq, n_embd] -> [n_seq, n_vocab]
    return x @ wte.T
Transformer Block (Previous Lecture)
Transformer Decoder
Prof Clark's Neural Networks course at Pomona College is ...

a great way to learn about the neural networks that are used in the field of neuroscience.

The course is designed to be a good introduction to the neural networks that are used in the field of neuroscience. It will cover the basics of neural networks, including the basics of the neural networks that are used in the field of neuroscience.

The course is designed to be a good introduction to the neural networks that are used in the field of neuroscience. It will cover the basics of neural networks,
Fine-Tuning

Scenario: use an LLM to answer questions about courses at the 5Cs

Attempt 1: Pre-trained Model

1. Ask ChatGPT (or any of the alternatives) questions

Issues?
Fine-Tuning

Scenario: *use an LLM to answer questions about courses at the 5Cs*

**Attempt 2: Custom Model**

1. Gather a *dataset* using *catalog information* and *reviews* on ASPC
2. Select an *existing LLM architecture*, call it PomGPT
3. Train PomGPT
4. Ask PomGPT questions

Issues?
Fine-Tuning

Scenario: use an LLM to answer questions about courses at the 5Cs

Attempt 3: Fine-tuned Model
1. Gather a dataset using catalog information and reviews on ASPC
2. Download a pre-trained ChatGPT model
3. Fine-tune the model and rename it PomGPT
4. Ask PomGPT questions

Issues?
Fine-Tuning

Scenario: *use an LLM to answer questions about courses at the 5Cs*

**Attempt 4: Fine-tuned Model and then use Reinforcement Learning**

1. Gather a dataset using catalog information and reviews on ASPC
2. Download a pre-trained ChatGPT model
3. Fine-tune the model and rename it PomGPT
4. Ask PomGPT questions and continue to gather human feedback (RLHF)

Issues?
RLHF

Supervised Learning Steps

Reinforcement Learning Steps
Resources

• Transformer models: an introduction and catalog — 2023 Edition - AI, software, tech, and people, not in that order... by X
• GPT in 60 Lines of NumPy | Jay Mody
• Fine-tuning 20B LLMs with RLHF on a 24GB consumer GPU
• Five years of GPT progress
• Transformer models: an introduction and catalog — 2023 Edition - AI, software, tech, and people, not in that order... by X