Contents

1 大模型与 AIGC

各种学习相关代码

https://github.com/daiwk/llms

1.1 小结

decoder 的并行化：https://zhuanlan.zhihu.com/p/368592551

- gpt1：transformer 的 decoder，参数量 117m（0.1b）
- gpt2：模型结构小改，增加数据，参数量变大（1.5b）
- gpt3：175b（1750 亿）参数，当参数量到达千亿时出现了『涌现』现象，发现可以 in-context learning
- Instructgpt：RLHF（sft–>rm–>ppo）
- gpt3.5：据说基本上等于 instructgpt
- gpt4：没公开细节，但听说效果很好，用起来也确实比 3.5 要好

1.2 llm 应用合辑

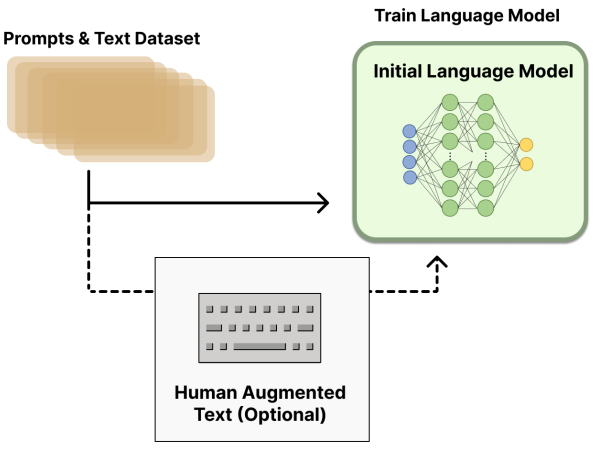
- ChatGPT 聚合站：https://hokex.com
- 游戏生成站：https://latitude.io/
- 家庭作业辅助站：https://ontimeai.com/
- 文字转语音站：https://www.resemble.ai/
- 艺术作画站：https://starryai.com/
- logo 制作站：https://www.logoai.com/
- ai 写作站：https://www.getconch.ai/
- 音乐制作站：https://soundraw.io/
- 声音模拟站：https://fakeyou.com/
- 一句话生成一段视频：https://runwayml.com/
- 文字转语音：https://murf.ai/

1.3 RLHF

https://huggingface.co/blog/zh/rlhf

- 预训练一个语言模型 (LM)；
- 聚合问答数据并训练一个奖励模型 (Reward Model，RM)，也叫偏好模型；
- 用强化学习 (RL) 方式微调 LM。

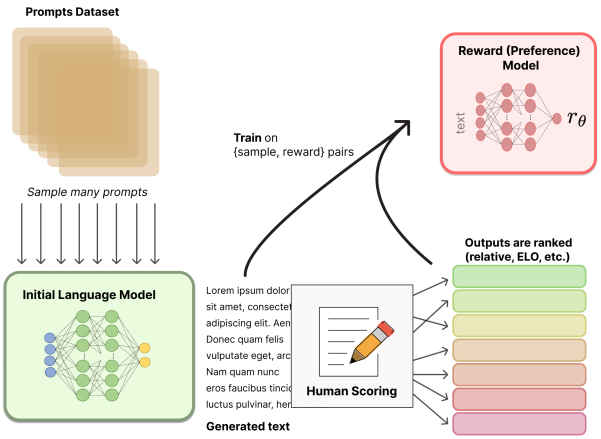
1.3.1 sft



- openai：instructGPT 使用小版本的 GPT-3，并对“更可取”（preferable）的人工生成文本微调
- Anthropic：1000w-520 亿参数的 transformer，并按“有用、诚实和无害”的标准在上下文线索上蒸馏原始 LM

- DeepMind：2800 亿的模型 Gopher

1.3.2 rm

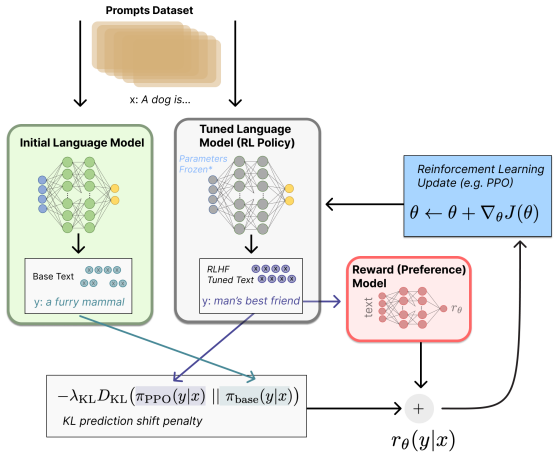


接收一系列文本并返回一个标量奖励，数值上对应人的偏好。我们可以用端到端的方式用 LM 建模，或者用模块化的系统建模 (比如对输出进行排名，再将排名转换为奖励) 。

- **模型选择**：RM 可以是另一个经过微调的 LM，也可以是根据偏好数据从头开始训练的 LM。Anthropic 提出了一种特殊的预训练方式，即用偏好模型预训练 (Preference Model Pretraining，PMP) 来替换一般预训练后的微调过程。因为前者被认为对样本数据的利用率更高。
- **训练文本**：RM 的提示 - 生成对文本是从预定义数据集中采样生成的，并用初始的 LM 给这些提示生成文本。Anthropic 的数据主要是通过 Amazon Mechanical Turk 上的聊天工具生成的，并在 Hub 上可用，而 OpenAI 使用了用户提交给 GPT API 的 prompt。
- **训练奖励数值**：人工对 LM 生成的回答进行排名。起初我们可能会认为应该直接对文本标注分数来训练 RM，但是由于标注者的价值观不同导致这些分数未经过校准并且充满噪音，通过排名可以比较多个模型的输出并构建更好的规范数据集，这些不同的排名结果将被归一化为用于训练的标量奖励值。

目前成功的 RLHF 系统使用了和生成模型具有不同大小的 LM，OpenAI 使用了 175B 的 LM 和 6B 的 RM，Anthropic 使用的 LM 和 RM 从 10B 到 52B 大小不等，DeepMind 使用了 70B 的 Chinchilla 模型分别作为 LM 和 RM

1.3.3 rl



直接微调整个 10B～100B+ 参数的成本过高，参考低秩自适应LoRA和 DeepMind 的Sparrow LM。目前多个组织找到的可行方案是使用策略梯度强化学习 (Policy Gradient RL) 算法、近端策略优化 (Proximal Policy Optimization，PPO) **微调初始 LM 的部分或全部参**

数。

- 策略 (policy)：一个接受提示并返回一系列文本 (或文本的概率分布) 的 LM
- 行动空间（action space）：LM 的词表对应的所有词元 (一般在 50k 数量级)
- 观察空间 (observation space)：是可能的输入词元序列，也比较大 (词汇量 ^ 输入标记的数量)
- 奖励函数：偏好模型和策略转变约束 (Policy shift constraint) 的结合。

ppo 确定的奖励函数如下：

- 提示 x 输入初始 LM 和当前微调的 LM，分别得到输出文本 y_1 和 y_2
- 将来自当前策略的文本传给 RM 得到标量奖励 r_θ
- 将两个模型的生成文本进行比较计算差异的惩罚项，一般是输出词分布间的 KL 散度的缩放，即 $r = r_\theta - \lambda r_{KL}$，

惩罚项的好处：+ 用于惩罚策略在每个训练 batch 中生成大幅偏离初始模型，以确保模型输出合理连贯的文本。+ 如果没有这一项，可能导致模型在优化中生成乱码文本，以愚弄奖励模型提供高奖励值。

根据 PPO，按当前 batch 的奖励进行优化。PPO 是置信域优化（TRO，Trust Region Optimization）算法，用梯度约束确保更新步骤不会破坏学习过程的稳定性。

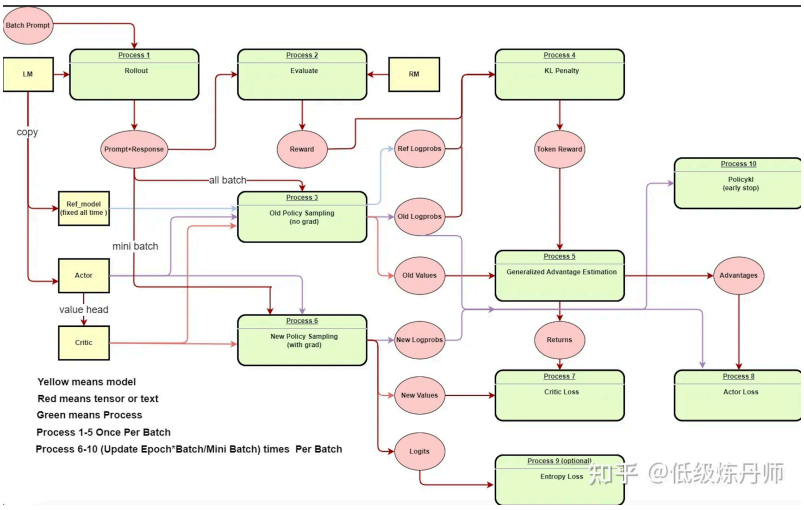
DeepMind 对 Gopher 用了类似的奖励设置，但用的是 A2C 来优化梯度。

1.3.3.1 概述

https://zhuanlan.zhihu.com/p/635757674

Fine-Tuning Language Models from Human Preferences

Secrets of RLHF in Large Language Models Part I: PPO



- Rollout and Evaluation：从 prompt 库里抽样，使用语言模型生成 response，然后使用奖励模型（Reward Model, RM）给出奖励得分。这个得分反映了生成的 response 的质量，比如它是否符合人类的偏好，是否符合任务的要求等。
- Make experience：收集了一系列的“经验”，即模型的行为和对应的奖励。这些经验包括了模型生成的 response 以及对应的奖励得分。这些经验将被用于下一步的优化过程。
- Optimization：使用收集到的经验来更新模型的参数。具体来说，我们使用 PPO 算法来调整模型的参数，使得模型生成的 response 的奖励得分能够增加。PPO 算法的一个关键特性是它尝试保持模型的行为不会发生太大的改变，这有助于保证模型的稳定性。

官方代码 example

```
from tqdm import tqdm

for epoch, batch in tqdm(enumerate(ppo_trainer.dataloader)):
    query_tensors = batch["input_ids"]
```

```
    #### Get response from SFTModel
    response_tensors = ppo_trainer.generate(query_tensors, **generation_kwargs)
    batch["response"] = [tokenizer.decode(r.squeeze()) for r in response_tensors]

    #### Compute reward score
    texts = [q + r for q, r in zip(batch["query"], batch["response"])]
    pipe_outputs = reward_model(texts)
    rewards = [torch.tensor(output[1]["score"]) for output in pipe_outputs]

    #### Run PPO step
    stats = ppo_trainer.step(query_tensors, response_tensors, rewards)
    ppo_trainer.log_stats(stats, batch, rewards)

#### Save model
ppo_trainer.save_model("my_ppo_model")
```

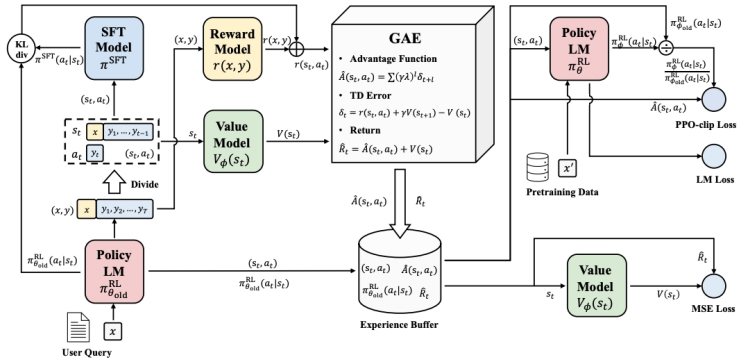


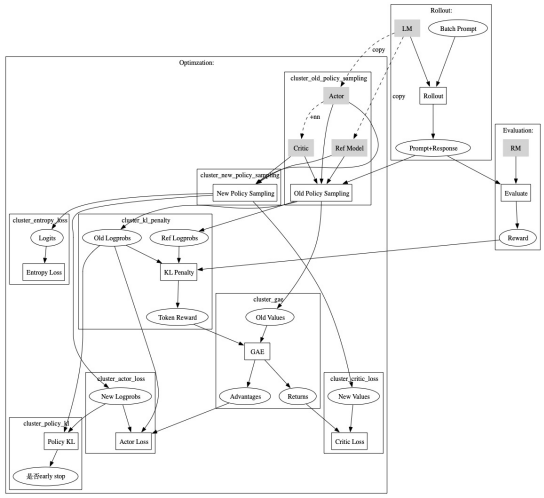
Figure 1: PPO workflow, depicting the sequential steps in the algorithm's execution. The process begins with sampling from the environment, followed by the application of GAE for improved advantage approximation. The diagram then illustrates the computation of various loss functions employed in PPO, signifying the iterative nature of the learning process and the policy updates derived from these losses.

- Rollout：根据策略（LM）生成轨迹（文本）。
  - 输入：Batch Prompt、LM
  - 输出：Prompt+Response
- Evaluate：对生成的轨迹进行评估（RM）。
  - 输入：Prompt+Response、RM
  - 输出：Reward
- Old Policy Sampling：计算并存储旧策略的概率、价值等值，
  - 输入：Ref_model、Actor、Critic、Prompt+Response
  - 输出：Ref Logprobs、Old Logprobs、Old Values
- KL Penalty：计算**当前策略**和**原始 LM** 之间的 KL 散度，用作对策略改变过快的惩罚项。
  - 输入：Ref Logprobs、Old Logprobs、Reward
  - 输出：Token Reward
- Generalized Advantage Estimation (GAE)：G。基于 old value(shape 是 (`batch_size`, `response_length`)) 和 reward 估计优势函数 A，它结合了所有可能的 n-step 进行 advantage 估计
  - 输入：Token Reward、Old Values
  - 输出：Advantages、Returns
- New Policy Sampling：
  - 输入 ref_model、actor、critic，从新的策略中采样概率等信息，
  - 输出 new logprobs、new values 和 logits，供 actor loss、critic loss 以及 entropy loss 用。
- Critic Loss：Critic 的目标是估计状态的价值函数，Critic loss 就是价值函数预测值和实际回报之间的差距。
  - 输入：New Values、Returns

  - 输出：critic 梯度更新
- Actor Loss：Actor 的目标是优化策略，Actor loss 就是基于优势函数的策略梯度。
  - 输入：Old Logprobs，New Logprobs、Advantages
  - 输出：actor 梯度更新
- Entropy Loss：为了增加探索性，通常会添加一个基于策略熵的正则项，它鼓励策略保持多样性。
  - 输入：Logits
  - 输出：entropy loss
- Policykl：这是对策略迭代过程的一个度量，它度量**新策略**和**旧策略**之间的差距。
  - 输入：Old Logprobs、New Logprobs
  - 输出：是否 early stop

在 PPO 中，策略优化的过程涉及到两个策略：一个是“旧的”策略，这是我们在开始每次优化迭代时使用的策略，另一个是“新的”策略，这是我们在优化过程中**不断更新**的策略。

自己整理重画的



1.3.3.2 actor & actor loss

Actor 是**策略**，它决定文本会被怎么样生成，是从**策略网络**拷贝来的模拟整个智能体在环境中行动的网络。

优势函数表示在给定的状态下采取某个行动比遵循当前策略的期望回报要好多少。

Actor Loss 如下，用重要性采样比较在**旧策略**和**新策略**下行动的概率（Old Logprobs，New Logprobs），然后将这个比值（也就是 Importance Sampling 的权重）与**优势函数 Advantages** 相乘，得到了对 Actor Loss 的一个估计。

$$L = \pi_{new}/\pi_{old} * A$$

```
# 计算新旧策略下概率的比值
ratio = torch.exp(logprobs - old_logprobs)

# 计算未截断的策略梯度损失
pg_losses = -advantages * ratio

# 计算截断的策略梯度损失
pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - self.config.cliprange, 1.0 + self.config.cliprange)

# 选择两者中较大的作为最终的策略梯度损失
pg_loss = masked_mean(torch.max(pg_losses, pg_losses2), mask)
```

```
# 计算因为截断导致策略梯度损失改变的比例
pg_clipfrac = masked_mean(torch.gt(pg_losses2, pg_losses).double(), mask)
```

1.3.3.3 critic & critic loss

critic 是专门用来预测 actor 轨迹**每一步价值**的网络，actor 上加几个线性层能够给每个 token 预测一个值。任务是估计状态的价值函数，也就是预测从当前状态开始，通过遵循某个策略，期望能得到的总回报。

Critic Loss 是最小化它的预测价值与实际回报之间的差距，常用 mse

通过最小化 Critic Loss，Critic 的预测能力会逐渐提升。因为 Critic 的预测结果会被用来**估计每个行动的优势（Advantage）**，这个优势值又会被用来计算策略的更新（Actor Loss）。

```
# 将价值函数的预测值裁剪到一个范围内
vpredclipped = clip_by_value(
            vpreds, values - self.config.cliprange_value, values + self.config.cliprange_value
        )

# 计算裁剪前和裁剪后的价值函数损失
vf_losses1 = (vpreds - returns) ** 2
vf_losses2 = (vpredclipped - returns) ** 2

# 最终的价值函数损失是裁剪前和裁剪后损失的最大值的平均值的一半
vf_loss = 0.5 * masked_mean(torch.max(vf_losses1, vf_losses2), mask)

# 计算裁剪操作实际发生的频率
vf_clipfrac = masked_mean(torch.gt(vf_losses2, vf_losses1).double(), mask)
```

1.3.3.4 Old Policy Sampling（无 bp）

是 **make experience** 的过程，计算并**存储**旧策略的概率、价值等值，来为后面更新的过程服务。

- Old Logprobs：从 “旧的” 策略 [即在这个 batch 数据中初始的 LM（initial actor）] 中计算每个 token 在旧的策略下的概率 Old Logprobs。
- Old Values：旧策略中每个**时间步**（每个 token 的预测结果）的价值，这个值由 critic 网络进行预测，critic 网络就是需要这个值的原因是 advantage 的计算依赖于 Old Values。
- Ref Logprobs：最最原始的 LM 对于每个时间步的概率预测，一般就是**固定不变的 gpt3**，计算这个值的目的是限制 actor 的更新，防止其偏离原始 gpt3 太远，他的实现在下一个步骤中。

```
all_logprobs, _, values, masks = self.batched_forward_pass(self.model, queries, responses, model_inputs)
ref_logprobs, _, _, _ = self.batched_forward_pass(self.ref_model, queries, responses, model_inputs)
```

1.3.3.5 KL Penalty

用于保证经过强化学习后的模型（新策略 actor）不会过于偏离原始预训练模型（ref model）。

```
# 初始化两个列表来分别存储奖励和非得分奖励
rewards, non_score_rewards = [], []

# 使用 zip 函数并行遍历输入的得分、对数概率、参考模型的对数概率以及 mask
for score, logprob, ref_logprob, mask in zip(scores, logprobs, ref_logprobs, masks):
    # 计算 KL 散度，即模型的对数概率与参考模型的对数概率之间的差值
    kl = logprob - ref_logprob

    # 计算非得分奖励，即 KL 散度乘以 KL 控制器值的负值
    non_score_reward = -self.kl_ctl.value * kl
    non_score_rewards.append(non_score_reward)
```

```
    # 复制非得分奖励为新的奖励
    reward = non_score_reward.clone()

    # 找到 mask 中最后一个非零元素的索引，这表示输入序列的实际长度
    last_non_masked_index = mask.nonzero()[-1]

    # 对于最后一个非 mask 部分的 token，其奖励是偏好模型的得分加上 KL 散度
    reward[last_non_masked_index] += score

    # 将计算的奖励添加到奖励列表中
    rewards.append(reward)

# 返回包含所有奖励的张量以及包含所有非得分奖励的张量
return torch.stack(rewards), torch.stack(non_score_rewards)
```

1.3.3.6 GAE

GAE 是一种多步优势估计方法。它通过引入一个权衡参数 λ，在**单步 TD 误差**和**多步 TD 误差**之间进行权衡，从而**减小估计的方差**，提高学习的稳定性。其中 σ_{t+l} 是时间步 $t+l$ 的 TD 误差。

$$A_t = \sum_{l=0}^{k-1} (\lambda\eta)^l \sigma_{t+l}$$

$$\sigma_{t+l} = r_{t+l+1} + \eta V(s_{t+l+1}) - V(s_{t+l})$$

```
# 从后往前遍历整个生成的序列
for t in reversed(range(gen_len)):
    # 计算下一个状态的价值，如果当前状态已经是最后一个状态，则下一个状态的价值为 0
    nextvalues = values[:, t + 1] if t < gen_len - 1 else 0.0

    # 计算 ，它是奖励加上衰减后的下一个状态的价值，然后减去当前状态的价值
    delta = rewards[:, t] + self.config.gamma * nextvalues - values[:, t]

    # 使用  更新 lastgaelam，这是 GAE 公式的一部分
    lastgaelam = delta + self.config.gamma * self.config.lam * lastgaelam

    # 将计算的优势值添加到优势值列表中
    advantages_reversed.append(lastgaelam)

# 将优势值列表反向并转换为张量
advantages = torch.stack(advantages_reversed[::-1]).transpose(0, 1)

# 计算回报值，它是优势值加上状态值
returns = advantages + values
```

1.3.3.7 New Policy Sampling（有 bp）

在**新的策略**（更新后的 actor）下对轨迹（文本）计算概率的过程，计算 Actor Loss，即策略梯度的损失。

Old Logprobs 是一次性一个 batch 的数据计算的，这是因为在一个 batch 中旧策略都是不变的；而 New Logprobs 是一个 mini batch 计算一次，这是因为新策略每个 mini batch 变一次。

1.3.3.8 entropy loss

一个策略的熵越大，意味着这个策略选择各个动作的概率更加“平均”。在 actor 的 loss 里加熵，使得策略的熵尽可能大，从而有更多机会探索可能带来更好奖励的文本轨迹。

```
entropy = -torch.sum(logits* torch.log(logits + 1e-9), dim=-1).mean()
```

新实现：

```
    pd = torch.nn.functional.softmax(logits, dim=-1)
    entropy = torch.logsumexp(logits, axis=-1) - torch.sum(pd * logits, axis=-1)
```

1.3.3.9 Policy kl

1.3.4 开源库

1.3.4.1 openai 的 lm-human-preferences(gpt2 的 finetune)

https://github.com/openai/lm-human-preferences

1.3.4.2 huggingface 的 TRL

https://github.com/huggingface/trl

1.3.4.3 CarperAI 的 trlx

https://github.com/CarperAI/trlx

1.3.4.4 allenai 的 RL4LMs

https://github.com/allenai/RL4LMs

1.4 LLM+ 推荐

推荐系统范式之争，LLM vs. ID?

1.5 NLP 大模型

1.5.1 nanogpt

简化版的 gpt，tiktoken：gpt2 中使用的开源分词工具，比 huggingface 的 tokenizer 快得多

```
import tiktoken
enc = tiktoken.get_encoding("gpt2")

# 字节对编码过程，我的输出是 [31373, 995]
encoding_res = enc.encode("hello world")
print(encoding_res)

# 字节对解码过程，解码结果：hello world
raw_text = enc.decode(encoding_res)
print(raw_text)
```

1.5.2 InstructGPT

OpenAI 魔改大模型，参数减少 100 倍！13 亿参数 InstructGPT 碾压 GPT-3

https://openai.com/blog/deep-reinforcement-learning-from-human-preferences/

Training language models to follow instructions with human feedback

1.5.3 Anthropic

Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback

Studying Large Language Model Generalization with Influence Functions

Measuring Faithfulness in Chain-of-Thought Reasoning

1.5.4 llama

LLaMA: Open and Efficient Foundation Language Models

参考代码：https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py

之前的工作考虑的是在训练预算有限的前提下，如何提升模型性能（2022 年 deepmind 的Training Compute-Optimal Large Language Models的 Chinchilla）,llama 考虑在预测时的预算。例如 chinchilla 是一个 10b 的模型在 200b 的 token 上训练，但其实一个 7b 的模型当用了 1T 的 token 后，性能仍在提升。LLama-13b 比 gpt3 在大多数 benchmark 上好，但 size 只有 1/10，在一个 GPU 上就能跑。

llama 只用公开数据训练，而 Chinchilla、PaLM、GPT-3 都有自己的未公开数据集。其他的 OPT、GPT-NeoX、BLOOM、GLM 虽然也只用公开数据集，但打不过 PaLM-62B 或者 Chinchilla

1.5.4.1 预训练数据

- English CommonCrawl(67%)：使用 CCNet pipeline，去重、用 fasttext 把非英文的页面删了，用 n-gram 把低质内容删了。此外，还训了一个线性模型，对页面进行分类：作为维基百科的引用 vs 随机采样的页面，最后把不属于引用这个类别的页面删了
- C4(15%)：与 CCNet 类似，主要区别在质量过滤是基于启发式的规则，如标点符号的存在，或者词数和句子数
- github(4.5%)：使用 Google BigQuery 里的公开 github 数据集，只用 Apache、BSD 和 MIT 证书的。低质判断是启发式规则，如字母数字占比、行的长度等，用正则删掉 head 等样式，最终以文件粒度进行去重。
- wikipedia(4.5%)：2022 年 6-8 月的数据，包括 20 种语言
- Gutenberg and Books3(4.5%)：两个书籍数据集，对有 90% 以上内容重复的书籍做去重。
- Arxiv(2.5%)：拿原始的 tex 文件，删掉 first section 之前的东西，还有一些注释、宏
- Stack Exchange(2%)：高质量的问答网站，按答案的分数排序

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

tokenizer：BPE，使用 sentencepiece 的实现。将所有 numbers 切成单个数字，回退到字节去处理未知的 utf8 字符（fallback to bytes to decompose unknown UTF-8 characters）

总共有 1.4T 的 token，对大部分训练数据，每个 token 在训练时只用了一次，除了维基和 book 大概用了两次。

附：gpt4 说：当我们说 “一个 token 只训练一次”，我们其实是在说在一个 epoch（一个完整遍历训练集的过程）中，我们只遍历一次完整的数据集。如果一个特定的 token 在数据集中出现多次，那么在一个 epoch 中，这个 token 就会被用来训练模型多次。

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

1.5.4.2 网络结构

- pre-normalization(gpt3)：提升训练**稳定性**，对每个子层的输入做 norm，而非输出。此外，使用的是 RMSNorm 函数 (Root mean square layer normalization)
- SwiGLU 激活函数 (PaLM)：Glu variants improve trans- former，把 PaLM 里的 $4d$ 改了 $2/34d$
- Rotary embeddings(GPTNeo)：删掉原来的绝对位置编码，加上 rotary positional embedding(RoPE)，网络的每一层都加，参考Roformer: En- hanced transformer with rotary position embedding

优化器：AdamW，cosine 学习率 schedule，最终学习率是最大学习率的 10%。0.1 的 weight decay 和 1.0 的 gradient cliping，使用 2000steps 的 warmup

1.5.4.3 训练加速

- 对 causal multi-head attention 加速：实现在http://github.com/facebookresearch/xformers中，降低内存使用和运行时间，参考self-attention does not need $o(n^2)$ memory，以及Flashattention: Fast and memory-efficient exact attention with io-awareness。思想是
  - 不存储 attention weights
  - 不计算被 mask 的 key/query 得分
- 减少 xxx：

1.5.5 llama2

Llama 2: Open Foundation and Fine-Tuned Chat Models

https://zhuanlan.zhihu.com/p/636784644

1.5.6 ChatGLM

ACL22 GLM: General Language Model Pretraining with Autoregressive Blank Infilling

iclr23 GLM-130B: An Open Bilingual Pre-trained Model

1.5.7 PALM-E

PaLM-E: An Embodied Multimodal Language Model

1.5.8 达摩院大模型技术交流

https://developer.aliyun.com/live/248332

ppt：链接 密码：5yyf

1.5.9 Google 的大规模稀疏模型设计

DESIGNING EFFECTIVE SPARSE EXPERT MODELS

代码：https://github.com/tensorflow/mesh/blob/master/mesh_tensorflow/transformer/moe.py

1.5.10 RETRO Transformer

参数量仅为 4%，性能媲美 GPT-3：开发者图解 DeepMind 的 RETRO

http://jalammar.github.io/illustrated-retrieval-transformer/

Improving language models by retrieving from trillions of tokens

1.5.11 WebGPT

WebGPT: Browser-assisted question-answering with human feedback

https://openai.com/blog/webgpt/

1.5.12 prompt

Fine-tune 之后的 NLP 新范式：Prompt 越来越火，CMU 华人博士后出了篇综述文章

1.5.13 ray-llm

https://github.com/ray-project/ray/releases/tag/ray-2.4.0

1.5.14 llm 相关汇总

llm 中文数据集

https://juejin.cn/post/7238921093553438779

简单综述

https://juejin.cn/post/7240022931078004797

1.5.15 llm for rec

Recommendation as Language Processing (RLP):A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5)

https://github.com/nancheng58/Awesome-LLM4RS-Papers

1.5.16 大模型的一些现象

1.5.16.1 重复生成

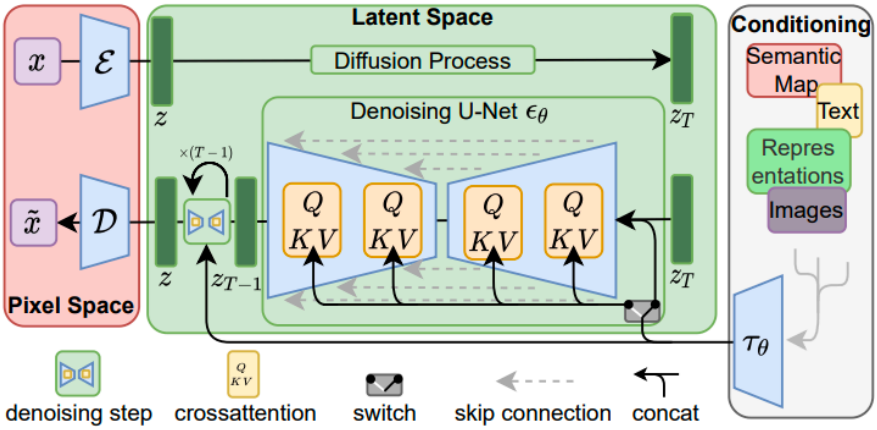
https://www.zhihu.com/question/616130636

https://mp.weixin.qq.com/s/cSwWapqFhxu9zafzPUeVEw

1.6 CV 大模型

1.6.1 stable diffusion

High-Resolution Image Synthesis with Latent Diffusion Models



输入图像，经过编码器得到 z，z 通过前向扩散不断加噪声得到 z_T（正向扩散）

输入条件，经过条件编码器 (原文是 BERT，到了 DALL-E2 就改成 CLIP 了) 得到 τ_θ

z_T 在 τ_θ 的指导下不断去噪（反向扩散），得到新的 z，再通过解码器得到最终生成的图像

1.7 多模态

【IEEE Fellow 何晓东 & 邓力】多模态智能论文综述：表示学习，信息融合与应用，259 篇文献带你了解 AI 热点技

Multimodal Intelligence: Representation Learning, Information Fusion, and Applications

BERT 在多模态领域中的应用

CV 领域：VisualBert, Unicoder-VL, VL-Bert, ViLBERT, LXMERT。

CLIP

1.8 其他

torch 里的 categorical 分布 (类别分布) https://blog.csdn.net/qq_37388085/article/details/127251550

https://zhuanlan.zhihu.com/p/59550457