【手搓大模型】从零微调GPT2

前文我们已经学会了从零写GPT-2代码，并从零开始训练。但众所周知，大模型的训练是极其昂贵的。在本文，我们不再自己训练模型，而是下载gpt-2的公开权重，直接load到我们自己的模型里，这样我们的模型就瞬间具备了与gpt-2相同的智能水平。

另外，即便是gpt-2，也只是原始的基础模型，仅能做文本补全，无法响应你的指令，我们将通过Fine tunning教会模型如何听从指令。

Understand Model Structure

加载公开权重的过程简单而枯燥，对最重要的是，要明白模型的结构，并确保自己的模型结构与公开的权重是配套的、一致的。

如下，可以详细查看模型的结构，包括每个参数的名字与维度：

代码块

```python
from gpt2_v2 import GPT2Model, GPT_CONFIG_124M, complete_text, generate_text_simple, tensor_to_text, text_to_tensor

GPT_CONFIG_124M.update({"qkv_bias": True})
model = GPT2Model(GPT_CONFIG_124M)
for name, param in model.named_parameters():
    print(name, param.shape)
```

> tok_emb.weight torch.Size([50257, 768])
>
> pos_emb.weight torch.Size([1024, 768])
>
> blocks.0.attn.W_Q.weight torch.Size([768, 768])
>
> blocks.0.attn.W_Q.bias torch.Size([768])
>
> blocks.0.attn.W_K.weight torch.Size([768, 768])
>
> blocks.0.attn.W_K.bias torch.Size([768])
>
> blocks.0.attn.W_V.weight torch.Size([768, 768])
>
> blocks.0.attn.W_V.bias torch.Size([768])
>
> blocks.0.attn.out_proj.weight torch.Size([768, 768])
>
> blocks.0.attn.out_proj.bias torch.Size([768])
>
> blocks.0.ff.layers.0.weight torch.Size([3072, 768])

> blocks.0.ff.layers.0.bias torch.Size([3072])
>
> blocks.0.ff.layers.2.weight torch.Size([768, 3072])
>
> blocks.0.ff.layers.2.bias torch.Size([768])
>
> blocks.0.ln1.weight torch.Size([768])
>
> blocks.0.ln1.bias torch.Size([768])
>
> blocks.0.ln2.weight torch.Size([768])
>
> blocks.0.ln2.bias torch.Size([768])
>
> ....[blocks 1-11 are omitted here]...............
>
> final_norm.weight torch.Size([768])
>
> final_norm.bias torch.Size([768])
>
> out_head.weight torch.Size([50257, 768])

Download and load GPT2 weights

下载与加载的过程，比较枯燥，几乎所有的工作都集中在参数处理与赋值上，就是找到我们模型中对应的参数在gpt2的权重里的名字，做映射，把正确的参数赋值到正确的位置上。

代码如下：

代码块

```python
from tqdm import tqdm
import urllib
import os
import json
from urllib.parse import urljoin
import tensorflow as tf
import numpy as np

def download_file(url, destination):
    def _attempt_download(download_url):
        with urllib.request.urlopen(download_url) as response:
            total_size = int(response.headers.get("Content-Length", 0))
            if os.path.exists(destination) and os.path.getsize(destination) == total_size:
                print(f"File already exists and is up-to-date: {destination}")
                return True

            with tqdm(total=total_size, unit="iB", unit_scale=True, desc=os.path.basename(download_url)) as pbar, \
                 open(destination, "wb") as f:
                for chunk in iter(lambda: response.read(1024), b""):
                    f.write(chunk)
                    pbar.update(len(chunk))
            return True

    try:
        if _attempt_download(url):
            return
    except Exception as e:
        print(f"Unexpected error: {e}")


def load_gpt2_params_from_tf_ckpt(ckpt_path, settings):
    # Initialize parameters dictionary with empty blocks for each layer
    params = {"blocks": [{} for _ in range(settings["n_layer"])]}

    # Iterate over each variable in the checkpoint
    for name, _ in tf.train.list_variables(ckpt_path):
        # Load the variable and remove singleton dimensions
        variable_array = np.squeeze(tf.train.load_variable(ckpt_path, name))

        # Process the variable name to extract relevant parts
        variable_name_parts = name.split("/")[1:]  # Skip the 'model/' prefix

        # Identify the target dictionary for the variable
        target_dict = params
        if variable_name_parts[0].startswith("h"):
            layer_number = int(variable_name_parts[0][1:])
            target_dict = params["blocks"][layer_number]

        # Recursively access or create nested dictionaries
        for key in variable_name_parts[1:-1]:
            target_dict = target_dict.setdefault(key, {})

        # Assign the variable array to the last key
        last_key = variable_name_parts[-1]
        target_dict[last_key] = variable_array

    return params


def download_and_load_gpt2(model_size, models_dir):
    allowed_sizes = {"124M", "355M", "774M", "1558M"}
    if model_size not in allowed_sizes:
        raise ValueError(f"Model size must be one of {allowed_sizes}")

    model_dir = os.path.join(models_dir, model_size)
    os.makedirs(model_dir, exist_ok=True)

    base_url = f"https://openaipublic.blob.core.windows.net/gpt-2/models/{model_size}/"

    filenames = [
        "checkpoint", "encoder.json", "hparams.json",
        "model.ckpt.data-00000-of-00001", "model.ckpt.index",
        "model.ckpt.meta", "vocab.bpe"
    ]

    for fname in filenames:
        dst = os.path.join(model_dir, fname)
        if os.path.exists(dst):
            print(f"Already exists: {fname}, skipping download.")
            continue
        primary = urljoin(base_url, fname)
        print(f"Downloading {fname} ...")
        download_file(primary, dst)

    tf_ckpt_path = tf.train.latest_checkpoint(model_dir)
    with open(os.path.join(model_dir, "hparams.json"), "r", encoding="utf-8") as f:
        settings = json.load(f)

    params = load_gpt2_params_from_tf_ckpt(tf_ckpt_path, settings)
    return settings, params
```

代码块

```python
settings, params = download_and_load_gpt2(model_size="124M", models_dir="gpt2")
print("Settings:", settings)
print("Params:", params.keys())
```

> Settings: {'n_vocab': 50257, 'n_ctx': 1024, 'n_embd': 768, 'n_head': 12, 'n_layer': 12}
>
> Params: dict_keys(['blocks', 'b', 'g', 'wpe', 'wte'])

需要注意的是，必须确保模型的结构与维度一致；另外我们的参数名字与gpt2权重中的不尽相同。下面是枯燥的copy参数过程，为了确保不出错，我们总是先检查参数的维度是否一致，如下：

代码块

```python
import torch
import numpy as np

def assign_(left, right):
    if right is None:
        raise ValueError("'right' cannot be None")
    right_tensor = torch.as_tensor(right, dtype=left.dtype, device=left.device)
    if right_tensor.numel() == 0:
        raise ValueError("'right' cannot be Empty")
    if left.shape != right_tensor.shape:
        raise ValueError(f"Shape mismatch: {left.shape} vs {right_tensor.shape}")
    with torch.no_grad():
        left.copy_(right_tensor)

def load_weights_into_gpt(gpt, params):
    assign_(gpt.pos_emb.weight, params["wpe"])
    assign_(gpt.tok_emb.weight, params["wte"])

    for b, (block, pblock) in enumerate(zip(gpt.blocks, params["blocks"])):
        # Attention QKV
        qw, kw, vw = np.split(pblock["attn"]["c_attn"]["w"], 3, axis=-1)
        qb, kb, vb = np.split(pblock["attn"]["c_attn"]["b"], 3, axis=-1)
        assign_(block.attn.W_Q.weight, qw.T)
        assign_(block.attn.W_K.weight, kw.T)
        assign_(block.attn.W_V.weight, vw.T)
        assign_(block.attn.W_Q.bias, qb)
        assign_(block.attn.W_K.bias, kb)
        assign_(block.attn.W_V.bias, vb)

        # Attention output projection
        assign_(block.attn.out_proj.weight, pblock["attn"]["c_proj"]["w"].T)
        assign_(block.attn.out_proj.bias,   pblock["attn"]["c_proj"]["b"])

        # Feedforward
        assign_(block.ff.layers[0].weight, pblock["mlp"]["c_fc"]["w"].T)
        assign_(block.ff.layers[0].bias,   pblock["mlp"]["c_fc"]["b"])
        assign_(block.ff.layers[2].weight, pblock["mlp"]["c_proj"]["w"].T)
        assign_(block.ff.layers[2].bias,   pblock["mlp"]["c_proj"]["b"])

        # LayerNorms
        assign_(block.ln1.weight, pblock["ln_1"]["g"])
        assign_(block.ln1.bias, pblock["ln_1"]["b"])
        assign_(block.ln2.weight, pblock["ln_2"]["g"])
        assign_(block.ln2.bias, pblock["ln_2"]["b"])

    assign_(gpt.final_norm.weight, params["g"])
    assign_(gpt.final_norm.bias, params["b"])
    assign_(gpt.out_head.weight,  params["wte"])
```

执行load权重，如下：

代码块

```
load_weights_into_gpt(model, params)
model.to("cpu")
model.eval()
```

现在，我们已经成功加载了gpt2的公开权重。

而检查参数加载的正确性非常简单，只需要再次运行我们的模型即可，如下：

代码块

```
import tiktoken

torch.manual_seed(123)
tokenizer = tiktoken.get_encoding("gpt2")
token_ids = generate_text_simple(
    model=model,
    idx=text_to_tensor("at the start of", tokenizer),
    max_new_tokens=15,
    context_size=GPT_CONFIG_124M["context_length"],
    top_k=25,
    temperature=1.4
)

print("Output text:\n", tensor_to_text(token_ids, tokenizer))
```

> Output text:
>
> at the start of an international series of events. You don't have to worry about who has

可见，生成出了较为通顺的句子，证明我们的模型加载公开权重是成功的。而如果模型加载错误，是无法输出合理的句子的。

Instruction Finetunning

截止目前，我们的gpt2模型还只是会补全文本，但并不能遵循指令。

Data prepare

下面，我们下载Alpaca数据集，用于指令训练。

可以通过这个地址快速下载：

代码块

```
!wget https://raw.githubusercontent.com/tatsu-lab/stanford_alpaca/main/alpaca_data.json
```

加载并查看数据如下：

代码块

```python
import json
import random

with open("alpaca_data.json", "r") as f:
    data = json.load(f)

random.seed(123)
data = random.sample(data, 1000)

```

代码块

```python
def format_input(entry):
    instruction = entry.get("instruction", "").strip()
    input_section = entry.get("input", "").strip()

    parts = [
        "Below is an instruction that describes a task. Write a response that appropriately completes the request.",
        "\n\n### Instruction:\n" + instruction,
    ]

    if input_section:
        parts.append("\n\n### Input:\n" + input_section)

    return "".join(parts)
```

代码块

```python
model_input = format_input(data[50])
desired_response = f"\n\n### Response:\n{data[50]['output']}"

print(model_input + desired_response)
```

代码块

```
Below is an instruction that describes a task. Write a response that
```

```
appropriately completes the request.

### Instruction:
Generate a general statement about the importance of empathy.

### Response:
Empathy is essential for fostering meaningful relationships, deepening
understanding between people, and building a more compassionate world.
```

后续我们会将包含instruction和response的完整文本输入给模型进行训练。

为了在个人pc上快速训练，我们仅随机选取1000条；格式如上，包含系统指令、instruction和对应的response。

Datasets

如下划分数据集，其中800条用于train，100条用于validate，100条用于test，如下：

代码块

```python
n = len(data)
train_data = data[:int(n * 0.80)]
test_data = data[int(n * 0.80):int(n * 0.90)]
val_data = data[int(n * 0.90):]
print("Training set length:", len(train_data))
print("Validation set length:", len(val_data))
print("Test set length:", len(test_data))
```

> Training set length: 800
>
> Validation set length: 100
>
> Test set length: 100

创建dataset，如下：

代码块

```python
import torch
from torch.utils.data import Dataset
from functools import partial

device = "cpu"  # or "cuda" if available

class InstructionDataset(Dataset):
    def __init__(self, data, tokenizer):
        self.encoded_texts = [
            tokenizer.encode(
                format_input(entry) + f"\n\n### Response:\n{entry['output']}"
            )
            for entry in data
        ]

    def __len__(self):
        return len(self.encoded_texts)

    def __getitem__(self, idx):
        return self.encoded_texts[idx]


def custom_collate_fn(
    batch,
    pad_token_id=50256,
    ignore_index=-100,
    allowed_max_length=None,
    device="cpu"
):
    max_len = min(
        max(len(seq) + 1 for seq in batch),
        allowed_max_length or float('inf')
    )

    input_tensors, label_tensors = [], []

    for seq in batch:
        seq = seq + [pad_token_id]
        padded = seq + [pad_token_id] * (max_len - len(seq))

        inputs = torch.tensor(padded[:-1], dtype=torch.long)
        labels = torch.tensor(padded[1:], dtype=torch.long)

        # Mask padding in labels except the first one
        pad_mask = (labels == pad_token_id).nonzero(as_tuple=True)[0]
        if len(pad_mask) > 1:
            labels[pad_mask[1:]] = ignore_index

        input_tensors.append(inputs)
        label_tensors.append(labels)

    return (
        torch.stack(input_tensors).to(device),
        torch.stack(label_tensors).to(device)
    )


```

```python
customized_collate_fn = partial(
    custom_collate_fn,
    device=device,
    allowed_max_length=1024
)
```

其中custom_collate_fn用于补齐训练样本，使得长度一致。

下面分别创建出用于train、validate、test的训练集：

代码块

```python
from torch.utils.data import DataLoader

torch.manual_seed(123)

train_dataset = InstructionDataset(train_data, tokenizer)
train_loader = DataLoader(train_dataset, batch_size=8, 
collate_fn=customized_collate_fn, shuffle=True,drop_last=True,num_workers=0)

val_dataset = InstructionDataset(val_data, tokenizer)
val_loader = DataLoader(val_dataset, batch_size=8, 
collate_fn=customized_collate_fn, shuffle=False,drop_last=False,num_workers=0)

test_dataset = InstructionDataset(test_data, tokenizer)
test_loader = DataLoader(test_dataset, batch_size=8, 
collate_fn=customized_collate_fn, shuffle=False,drop_last=False,num_workers=0)
```

其中指定batch_size=8;

我们可以拿出val_data中的一条，测试下当前的模型，如下：

代码块

```python
model.eval()
torch.manual_seed(123)

input_text = format_input(val_data[3])
token_ids = generate_text_simple(
    model=model,
    idx=text_to_tensor(input_text, tokenizer),
    max_new_tokens=50,
    context_size=1024,
    eos_id=50256,
)
generated_text = tensor_to_text(token_ids, tokenizer)
print(generated_text)
```

代码块

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Describe the origins and history of the Internet.

### Response:

Describe the origin and history of the Internet.

### Response:

Describe the origin and history of the Internet.

### Response:

Describe the origin and history of the
```

可见，模型并不会遵从指令，只是在无意义的重复指令。

我们可以查看下当前的模型在训练集和验证集上的loss，如下：

代码块

```python
from gpt2_v2 import loss_loader

model.to(device)

torch.manual_seed(123)

with torch.no_grad():
    train_loss = loss_loader(train_loader, model, device, num_batches=5)
    val_loss = loss_loader(val_loader, model, device, num_batches=5)

print("Training loss:", train_loss)
print("Validation loss:", val_loss)
```

> Training loss: 3.5710490226745604
>
> Validation loss: 3.468023490905762

Train as normal

而Finetune的过程与通常的train类似，代码如下：

代码块

```
import torch
import time
from gpt2_v2 import train_model_simple, build_tokenizer

torch.manual_seed(123)
torch.set_num_threads(12)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5, weight_decay=0.1)

start_time = time.time()

# FineTune the model
num_epochs = 2
train_losses, val_losses, tokens_seen = train_model_simple(
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    optimizer=optimizer,
    device=device,
    num_epochs=num_epochs,
    eval_freq=5,
    eval_iter=5,
    start_context=format_input(val_data[3]),
    tokenizer=build_tokenizer()
)

elapsed = (time.time() - start_time) / 60
print(f"Training completed in {elapsed:.2f} minutes.")
```

代码块

```
Ep 1 (Step 000005): Train loss 2.725, Val loss 2.635, Tokens seen: 6424
Ep 1 (Step 000010): Train loss 2.191, Val loss 2.175, Tokens seen: 14624
Ep 1 (Step 000015): Train loss 2.028, Val loss 2.044, Tokens seen: 21104
Ep 1 (Step 000020): Train loss 2.043, Val loss 1.969, Tokens seen: 28240
Ep 1 (Step 000025): Train loss 1.945, Val loss 1.948, Tokens seen: 36288
Ep 1 (Step 000030): Train loss 1.764, Val loss 1.918, Tokens seen: 44584
Ep 1 (Step 000035): Train loss 1.882, Val loss 1.893, Tokens seen: 53184
Ep 1 (Step 000040): Train loss 1.854, Val loss 1.895, Tokens seen: 60104
Ep 1 (Step 000045): Train loss 1.964, Val loss 1.880, Tokens seen: 72128
Ep 1 (Step 000050): Train loss 1.807, Val loss 1.854, Tokens seen: 81440
Ep 1 (Step 000055): Train loss 1.738, Val loss 1.857, Tokens seen: 88864
Ep 1 (Step 000060): Train loss 1.783, Val loss 1.859, Tokens seen: 97120
Ep 1 (Step 000065): Train loss 1.770, Val loss 1.855, Tokens seen: 104936
Ep 1 (Step 000070): Train loss 1.792, Val loss 1.845, Tokens seen: 112192
Ep 1 (Step 000075): Train loss 1.819, Val loss 1.834, Tokens seen: 119536
Ep 1 (Step 000080): Train loss 1.771, Val loss 1.830, Tokens seen: 126736
Ep 1 (Step 000085): Train loss 1.755, Val loss 1.831, Tokens seen: 135560
Ep 1 (Step 000090): Train loss 1.626, Val loss 1.829, Tokens seen: 144096
Ep 1 (Step 000095): Train loss 1.659, Val loss 1.816, Tokens seen: 154272
Ep 1 (Step 000100): Train loss 1.681, Val loss 1.808, Tokens seen: 162040
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Describe the origins and history of the Internet.

### Response:
The Internet was invented by the internet community in the early 1800s
💾 Checkpoint saved: checkpoints/checkpoint_epoch1.pth
Ep 2 (Step 000105): Train loss 1.760, Val loss 1.824, Tokens seen: 168504
Ep 2 (Step 000110): Train loss 1.507, Val loss 1.811, Tokens seen: 174832
Ep 2 (Step 000115): Train loss 1.556, Val loss 1.825, Tokens seen: 182232
Ep 2 (Step 000120): Train loss 1.597, Val loss 1.816, Tokens seen: 188792
Ep 2 (Step 000125): Train loss 1.524, Val loss 1.807, Tokens seen: 197440
Ep 2 (Step 000130): Train loss 1.629, Val loss 1.827, Tokens seen: 205488
Ep 2 (Step 000135): Train loss 1.613, Val loss 1.806, Tokens seen: 213760
Ep 2 (Step 000140): Train loss 1.543, Val loss 1.813, Tokens seen: 222032
Ep 2 (Step 000145): Train loss 1.599, Val loss 1.820, Tokens seen: 230560
Ep 2 (Step 000150): Train loss 1.519, Val loss 1.817, Tokens seen: 240576
Ep 2 (Step 000155): Train loss 1.450, Val loss 1.818, Tokens seen: 248984
Ep 2 (Step 000160): Train loss 1.617, Val loss 1.799, Tokens seen: 256312
Ep 2 (Step 000165): Train loss 1.541, Val loss 1.808, Tokens seen: 264560
Ep 2 (Step 000170): Train loss 1.605, Val loss 1.794, Tokens seen: 271744
Ep 2 (Step 000175): Train loss 1.453, Val loss 1.801, Tokens seen: 280280
Ep 2 (Step 000180): Train loss 1.524, Val loss 1.806, Tokens seen: 286704
Ep 2 (Step 000185): Train loss 1.457, Val loss 1.791, Tokens seen: 297816
Ep 2 (Step 000190): Train loss 1.414, Val loss 1.794, Tokens seen: 303968
Ep 2 (Step 000195): Train loss 1.484, Val loss 1.799, Tokens seen: 313272
Ep 2 (Step 000200): Train loss 1.499, Val loss 1.804, Tokens seen: 320096
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Describe the origins and history of the Internet.

### Response:
The Internet was invented by the internet community in the early 1800s
💾 Checkpoint saved: checkpoints/checkpoint_epoch2.pth
```

```
🎉 Training complete. Final model saved: checkpoints/final_model.pth
Training completed in 11.32 minutes.
```

可见，仅仅通过1-2个epoch，模型就学会了遵从指令。

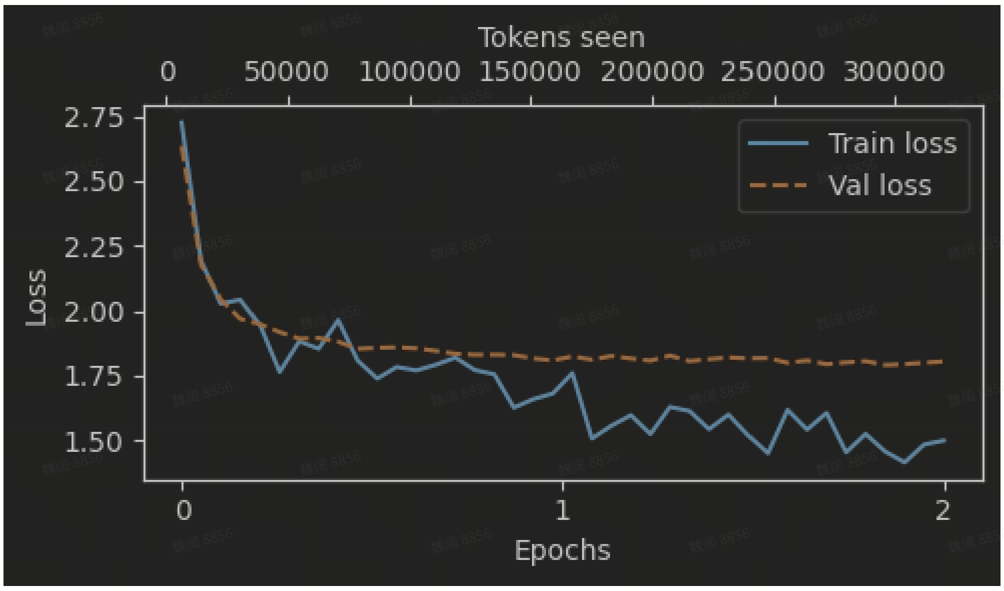
因为仅为了演示，数据量非常小，此处仅仅跑了2个epoch；有兴趣的可以增大数据量试试。

我们可以可视化看看loss的变化情况

代码块

```python
from gpt2_v2 import plot_losses

epochs_tensor = torch.linspace(0, num_epochs, len(train_losses))
plot_losses(epochs_tensor, tokens_seen, train_losses, val_losses)
```



可见，loss在训练集上快速下降；但在测试集上下降不大；这也符合预期，毕竟为了演示目的，所用数据量太少。有兴趣的可以增大数据量试试。

Save model

对于训练好的模型，我们可以保存下来，便于下次加载重现，如下：

简单点就一行代码：

代码块

```python
file_name = "gpt2-124M-sft.pth"
torch.save(model.state_dict(), file_name)
print(f"Model saved as {file_name}")
```

而复杂点，我们可以保存其他描述性的参数，特别适用于模型训练过程中，如下：

代码块

```python
# Save final model
final_path = os.path.join(save_dir, "final_model.pth")
torch.save({
    'epoch': num_epochs,
    'step': step,
    'tokens_seen': tokens_seen,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'train_losses': train_losses,
    'val_losses': val_losses,
    'tokens_seen_track': tokens_seen_track,
}, final_path)
print(f"🎉 Training complete. Final model saved: {final_path}")
```

存储模型的过程有点类似工程上的序列化与反序列化。可见，模型本质上就是神经网络结构+各层的参数。

而在较大数据量的训练中，有时候训练会意外中断，而为了使得训练结果不丢失，可以周期性地，如每个epoch结束，都保存下模型参数；这样便于中断后恢复，继续训练，节约时间和成本。

Evaluate model

模型的效果到底如何，除了可以人工抽查、人工评估之外，我们还可以借助其他更强大的模型，来给我们的小模型评估打分。

Generate response

首先，我们跑test_data，拿测试问题输入模型，生成所有的答复，如下：

代码块

```python
from tqdm import tqdm
import json

def generate_response(entry, model):
    input_text = format_input(entry)
    token_ids = generate_text_simple(
        model=model,
        idx=text_to_tensor(input_text, tokenizer),
        max_new_tokens=35,
        context_size=1024,
        eos_id=50256,
    )
    generated_text = tensor_to_text(token_ids, tokenizer)
    response = generated_text[len(input_text):].replace("### Response:", 
"").strip()
    return response

# Generate and attach responses
for entry in tqdm(test_data, desc="Generating responses"):
    entry["model_response"] = generate_response(entry, model)

# Save to file
with open("instruction-data-with-response.json", "w") as f:
    json.dump(test_data, f, indent=4)
```

Run Ollama and Llama3

接下来，我们下载并打开ollama；Ollama 是一个开源工具，用于在本地计算机上运行、部署和管理大型语言模型；官方支持了DeepSeek-R1, Qwen 3, Llama 3.3, Qwen 2.5‑VL, Gemma 3等模型的本地部署与运行。使用过程非常简单，网上大量教程，此处不再赘述。

以下是简单地判断Ollama运行状态，以及本地发送http请求到llama3模型的代码：

代码块

```python
import psutil

def is_process_running(name_substr: str) -> bool:
    """Check if any running process contains the given substring in its 
name."""
    return any(name_substr.lower() in (proc.info["name"] or "").lower()
               for proc in psutil.process_iter(["name"]))

if not is_process_running("ollama"):
    raise RuntimeError("❌ Ollama not running. Please launch it before 
proceeding.")

print("✅ Ollama is running.")
```

代码块

```python
import json
import urllib.request

def query_model(
    prompt: str,
    model: str = "llama3",
    url: str = "http://localhost:11434/api/chat",
    seed: int = 123,
    temperature: float = 0.0,
    num_ctx: int = 2048
) -> str:
    """Send a prompt to a local chat model and return the generated response."""

    data = {
        "model": model,
        "messages": [{"role": "user", "content": prompt}],
        "options": {
            "seed": seed,
            "temperature": temperature,
            "num_ctx": num_ctx
        }
    }

    request = urllib.request.Request(
        url,
        data=json.dumps(data).encode("utf-8"),
        method="POST",
        headers={"Content-Type": "application/json"}
    )

    response_text = []

    try:
        with urllib.request.urlopen(request) as response:
            for line in response:
                line = line.decode("utf-8").strip()
                if line:
                    message_chunk = json.loads(line)
                    content = message_chunk.get("message", {}).get("content", "")
                    response_text.append(content)
    except Exception as e:
        raise RuntimeError(f"Failed to query model: {e}")

    return "".join(response_text)
```

上述代码非常简单，就是拼装参数，发送http请求到本地的llama3模型上。

此处，我们选择llama3作为我们的裁判模型，因为gpt2有1.5B参数，而llama3有8B参数，性能上有显著提升。

我们可以简单测试下上述代码，问llama3模型一个简单的问题，如下：

```python
result = query_model("What do Llamas eat?", "llama3")
print(result)
```

代码块

```
Llamas are herbivores, which means they primarily feed on plant-based foods. Their diet typically consists of:

1. Grasses: Llamas love to graze on various types of grasses, including tall grasses, short grasses, and even weeds.
2. Hay: High-quality hay, such as alfalfa or timothy hay, is a staple in a llama's diet. They enjoy the sweet taste and texture of fresh hay.
3. Grains: Llamas may receive grains like oats, barley, or corn as part of their daily ration. However, it's essential to provide these grains in moderation, as they can be high in calories.
4. Fruits and vegetables: Llamas enjoy a variety of fruits and veggies, such as apples, carrots, sweet potatoes, and leafy greens like kale or spinach.
5. Minerals: Llamas require access to mineral supplements, which help maintain their overall health and well-being.

In the wild, llamas might also eat:

1. Leaves: They'll munch on leaves from trees and shrubs, including plants like willow, alder, and birch.
2. Bark: In some cases, llamas may eat the bark of certain trees, like aspen or cottonwood.
3. Mosses and lichens: These non-vascular plants can be a tasty snack for llamas.

In captivity, llama owners typically provide a balanced diet that includes a mix of hay, grains, and fruits/vegetables. It's essential to consult with a veterinarian or experienced llama breeder to determine the best feeding plan for your llama.
```

可见，llama3在本地运行正常，且其输出质量还是相当高的。

Evaluate by scores

最后，我们写个prompt，让llama3给我们的gpt2生成的response进行打分，如下：

代码块

```python
from tqdm import tqdm

def generate_model_scores(data, response_key="model_response", model="llama3"):
    """Generate integer scores (0-100) for model responses using LLM evaluation."""
    scores = []

    for entry in tqdm(data, desc="Scoring entries"):
        prompt = (
            "Given the input below, the correct output, and the model's response, "
            "score the model's response on a scale from 0 to 100, where 100 is the best.\n\n"
            f"### Input:\n{format_input(entry)}\n\n"
            f"### Expected Output:\n{entry['output']}\n\n"
            f"### Model Response:\n{entry.get(response_key, '').strip()}\n\n"
            "### Respond with the integer number only."
        )

        try:
            score_str = query_model(prompt, model=model).strip()
            score = int(score_str)
            scores.append(score)
        except ValueError:
            print(f"[Warning] Invalid score format: {score_str!r}")
        except Exception as e:
            print(f"[Error] Scoring failed for entry: {e}")

    return scores
```

代码块

```python
scores = generate_model_scores(test_data)
print(f"Number of scores: {len(scores)} of {len(test_data)}")
print(f"Average: {sum(scores)/len(scores):.2f}, Max: {max(scores)}, Min: {min(scores)}")
```

> Number of scores: 85 of 100
>
> Average: 56.62, Max: 85, Min: 0

排除掉一些输出格式错误，平均得分56；虽然没及格，但是不算太差；至少说明模型比最开始的胡说八道进步不小。

至此，我们已经完整地学会了如何构建gpt2代码、训练模型和微调模型的必备技能。文章仅是演示用途，可以自行扩大数据集，放到更高性能的GPU上训练、微调，体验更多大模型的乐趣。