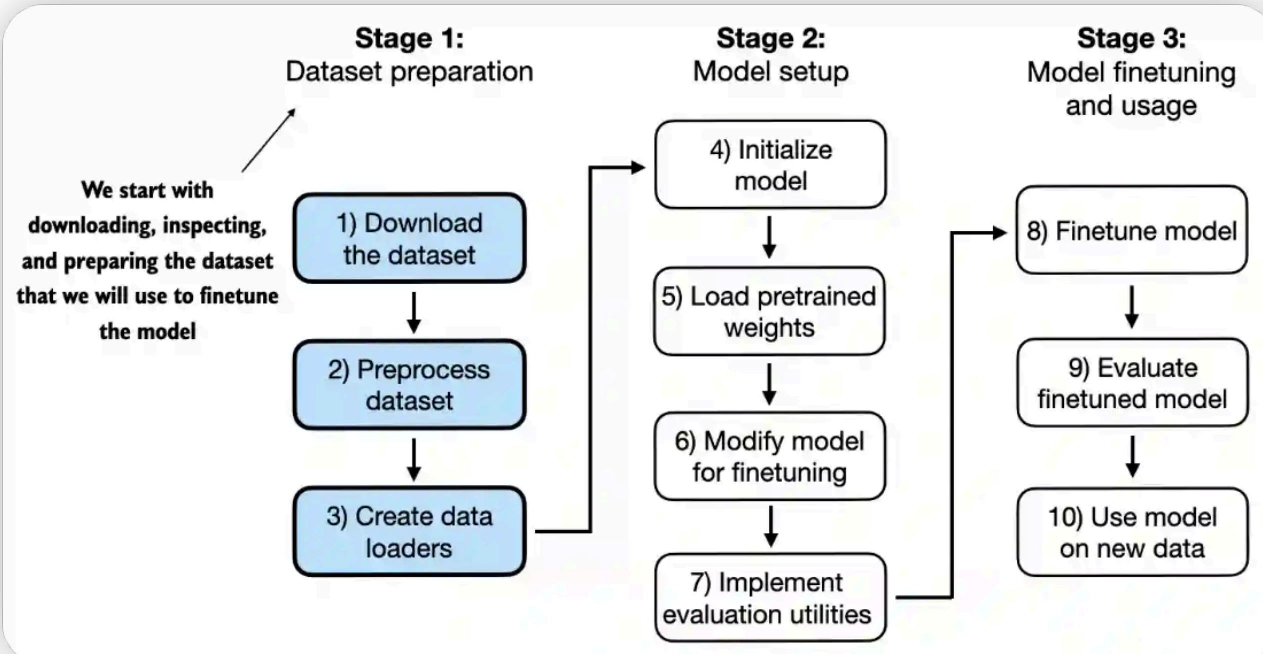
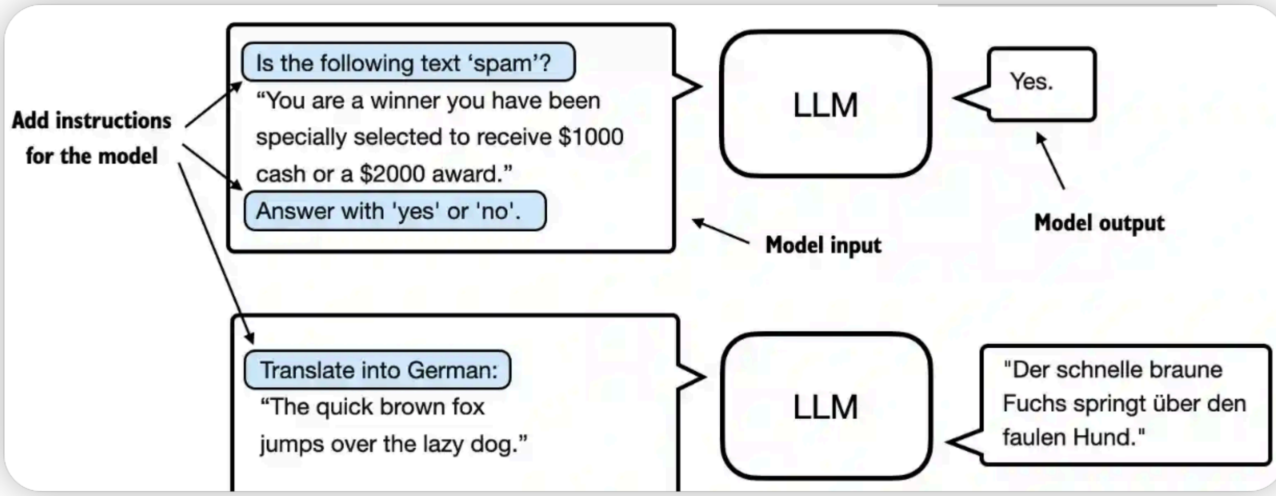
> LLM的预训练-微调范式，当基础模型预训练完成后，可以根据具体的应用场景和任务需求，在特定的小型有标签数据集上对其进行微调。

本文主要是利用5-预训练得到的模型（一次生成一个单词），载入参数。然后修改模型最后一层，再次训练实现分类功能。包含以下全部步骤，从 1) 下载数据集到 10) 使用模型。



6 - 1 微调的类别

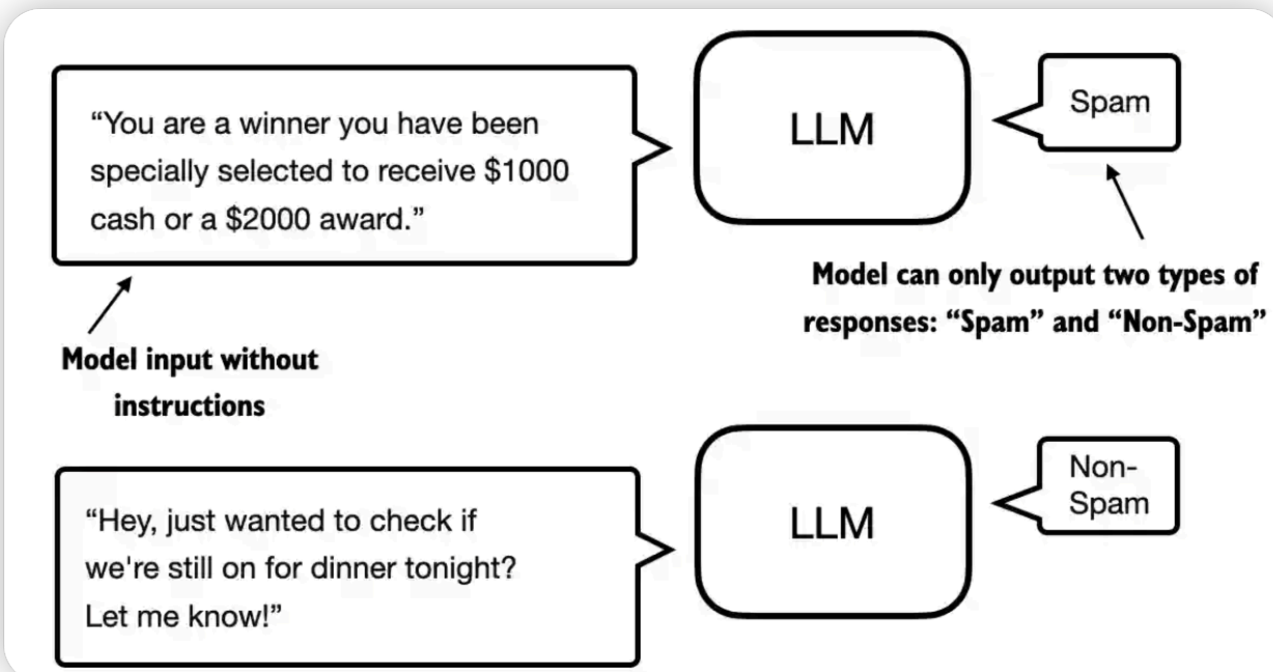
指令微调



在第七章讲。

分类微调

分类微调通常只能预测训练时看到的类别，指令微调可执行更多任务。



6 - 2 准备数据集

使用一个包含垃圾 spam 4825条和非垃圾邮件 ham 747条的数据集。

- 数据集类别不平衡，需简单的平衡处理

```
# 1 - 读取为 DataFrame

import pandas as pd
df = pd.read_csv(data_file_path, sep="\t", header=None, names=
["Label", "Text"])
```

```
# 2 - 简单的平衡

def create_balanced_dataset(df):
    # 选择Label为spam的列，再shape[0]得有多少行
    num_spam = df[df["Label"] == "spam"].shape[0]
    # 抽取num_spam个Label为ham的列
    ham_subset = df[df["Label"] == "ham"].sample(num_spam,
random_state=123)
    # 合并
    balanced_df = pd.concat([ham_subset, df[df["Label"] ==
"spam"]])
    return balanced_df
balanced_df = create_balanced_dataset(df)
```

```
# 3 - 特征工程: ham to 0,  spam to 1

balanced_df["Label"] = balanced_df["Label"].map({
    "ham": 0,
    "spam": 1
})
```

```
# 4 - 划分训练，验证，测试集

def random_split(df, train_frac, validation_frac):
    df = df.sample(frac=1, random_state=123).reset_index(drop=True)

    train_end = int(len(df) * train_frac)
    validation_end = train_end + int(len(df) * validation_frac)
```

```python
    train_df = df[:train_end]
    validation_df = df[train_end:validation_end]
    test_df = df[validation_end:]

    return train_df, validation_df, test_df

train_df, validation_df, test_df = random_split(balanced_df, 0.7,
0.1)

train_df.to_csv("train.csv", index=None)
validation_df.to_csv("validation.csv", index=None)
test_df.to_csv("test.csv", index=None)
```

6 - 3 创建数据加载器

Dataset

> text长度不同，需截断或补长，这里选择全部补长到最长 text 长度，使用 `<|endoftext|>` 作为补充token

```python
# 找到最长长度并补充其他text

import torch
from torch.utils.data import Dataset

class SpamDataset(Dataset):
    """找到最长长度并补充其他text

    Args:
        Dataset : torch的数据集类型
    """
    def __init__(self, csv_file, tokenizer, max_length=None,
pad_token_id=50256):
        self.data = pd.read_csv(csv_file)
        self.encoded_texts = [
            tokenizer.encode(text) for text in self.data["Text"]
```

```
        ]

        if max_length is None:
            self.max_length = self._longest_encoded_length()
        else:
            self.max_length = max_length
            self.encoded_texts = [
                encoded_text[:self.max_length]
                for encoded_text in self.encoded_texts
            ]
        self.encoded_texts = [
            encoded_text + [pad_token_id] * (self.max_length -
len(encoded_text))
            for encoded_text in self.encoded_texts
        ]

    def __getitem__(self, index):
        encoded = self.encoded_texts[index]
        label = self.data.iloc[index]["Label"]
        return (
            torch.tensor(encoded, dtype=torch.long),
            torch.tensor(label, dtype=torch.long)
        )

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

    def _longest_encoded_length(self):
        max_length = 0
        for encoded_text in self.encoded_texts:
            encoded_length = len(encoded_text)
            if encoded_length > max_length:
                max_length = encoded_length
        return max_length
```

DataLoader

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

```
num_workers = 0
batch_size = 8

torch.manual_seed(123)

train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=num_workers,
    drop_last=True,
)

val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    drop_last=False,
)

test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    drop_last=False,
)
```

6 - 4 使用预训练权重初始化模型

> 实现上述步骤图的4）初始化模型和5）加载预训练权重。

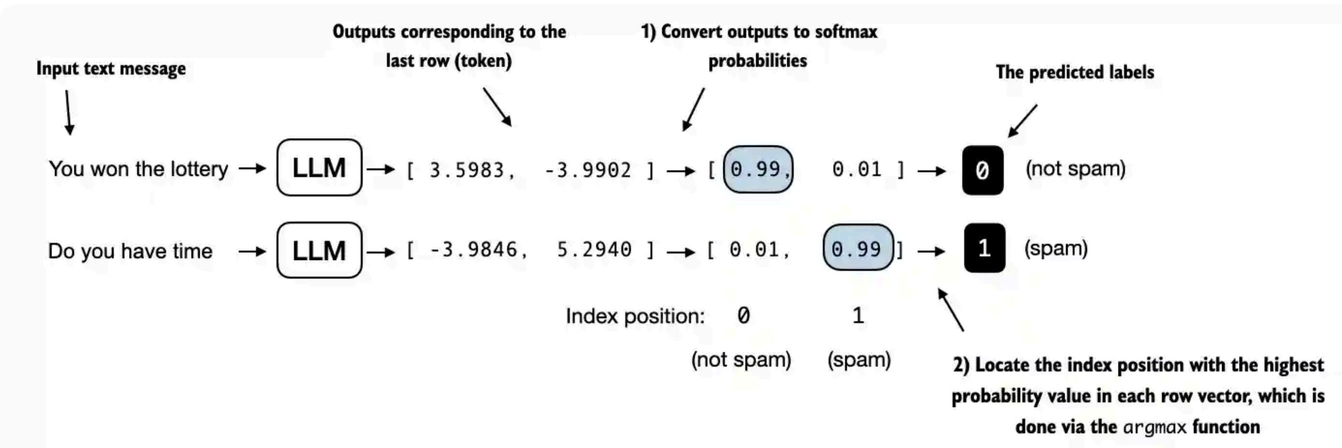
- 将开源的，预训练模型的参数设置为模型参数。
  书中的模型url文件已不存在。选择从 Hugging Face Hub 加载权重的方法。

6 - 5 添加分类头 - 修改模型

```
model.out_head =
torch.nn.Linear(in_features=BASE_CONFIG["emb_dim"],
out_features=num_classes)
```

上面代码实现了将 `(out_head): Linear(in_features=768, out_features=50257, bias=False)` 修改为 `(out_head): Linear(in_features=768, out_features=2, bias=True)`

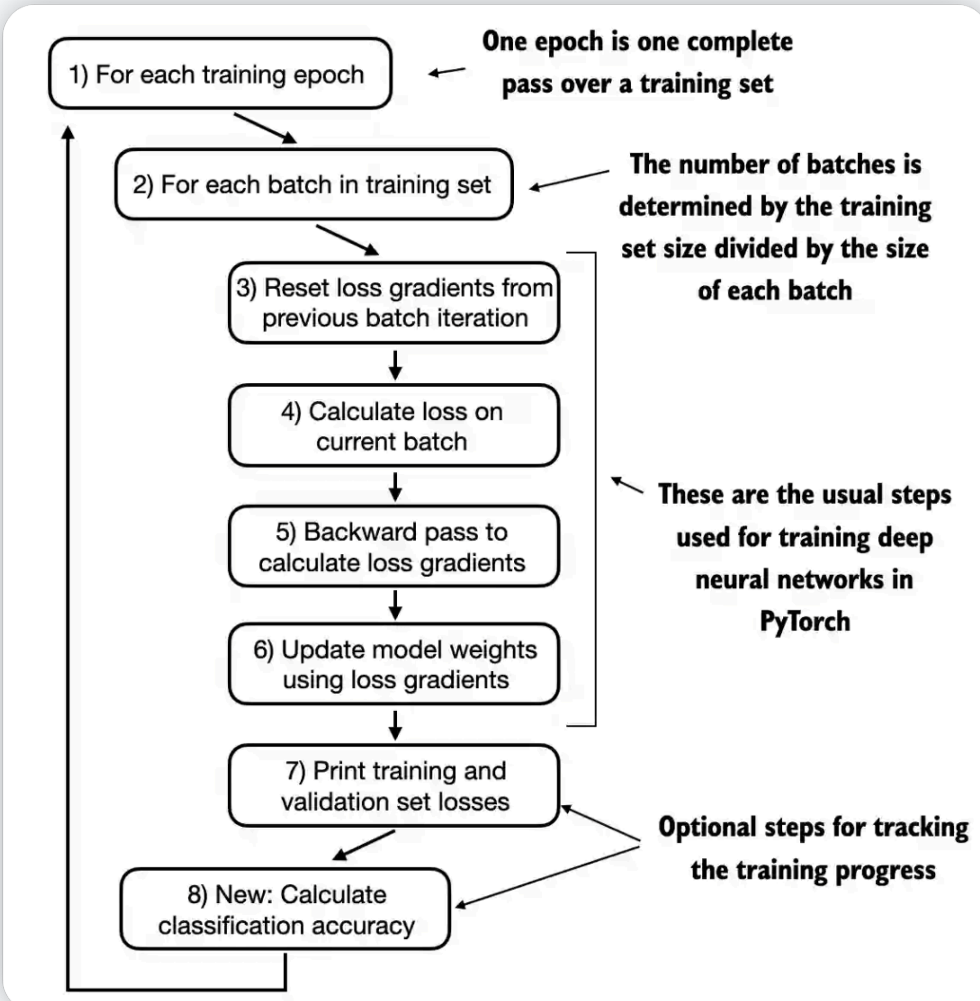
然后使用 Softmax函数 将输出向量转换为标签概率。



6 - 6 计算分类损失和准确率

准确率不是可微函数，我们选择交叉熵函数作为损失函数。和预训练计算损失函数如出一辙。

6 - 7 根据监督数据微调模型

> 与预训练的模型训练步骤类似。
>
> 
>

本ipynb是选用了GPT模型，outputs的最后一个token（`model(input_batch)[:, -1, :]`）来微调。也可以选择其他的来微调。

6 - 8 使用大模型作为垃圾邮件分类器

```
def classify_review(text, model, tokenizer, device,
max_length=None, pad_token_id=50256):
    model.eval()
    input_ids = tokenizer.encode(text)
    supported_context_length = model.pos_emb.weight.shape[1]

    input_ids = input_ids[:min(max_length,
```

```
supported_context_length)]
    input_ids += [pad_token_id] * (max_length - len(input_ids))
    input_tensor = torch.tensor(input_ids,
device=device).unsqueeze(0)

    with torch.no_grad():
        logits = model(input_tensor)[:, -1, :]
    predicted_label = torch.argmax(logits, dim=-1).item()

    return "spam" if predicted_label == 1 else "not spam"
```

模型的保存与复用

```
# 保存模型
torch.save(model.state_dict(), "review_classifier.pth")

# 加载模型
model_state_dict = torch.load("review_classifier.pth",
map_location=device, weights_only=True)
model.load_state_dict(model_state_dict)
```