

文本生成

1 背景介绍

大语言模型 (Large Language Model) 近年来发展迅猛, 以 ChatGPT 为代表的对话系统已经展现了强大的文本生成能力, 并且逐渐成为我们的工作生活中的日常工具。本次作业将介绍大语言模型背后的一些基础技术, 并且以对《Harry Potter》系列小说的仿写作为例子展示如何使用这些技术。通过本次作业, 你会学到

- 如何将文本转变为可使用的训练数据;
- Transformer 模型是怎样构建和使用的, Transformer 中的不同设计有什么作用;
- 自回归方法是如何完成的;
- 一个《Harry Potter》小说段落生成器。

1.1 自回归方法

传统的模型训练方式通常要求输入数据有对应的人工标注, 模型接受输入数据, 预测对应的数据标注, 即监督学习。在文本生成这个任务中, 我们可以利用文本数据的特点, 在生成时使用目前已有的文本去预测下一个词符, 然后将新的词符加入已有的文本, 继续生成下一个词符; 在训练时, 我们就可以将输入文本后的词符作为标注来训练数据, 这种方法被称为自回归方法。

具体来说, 假设文本序列一共有 n 个词符 x_1, \dots, x_n , 如果输入为前 $i-1$ 个词符 x_1, \dots, x_{i-1} , 那么预测目标就是 x_i , 因此自回归方法将这串文本的联合分布建模为

$$p(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}). \quad (1)$$

训练时, 希望最大化这串文本的似然函数, 也就是最大化

$$L = \sum_{i=1}^n \log p(x_i | x_1, \dots, x_{i-1}). \quad (2)$$

1.2 作业说明

我们提供了完成本次作业需要的大部分代码框架, 你需要按照本文档的说明完成其中 `dataset.py`、`model.py` 和 `generate.py` 三个文件里的部分功能, 并且根据代码和实验结果回答问题, 撰写报告, 最后请将填充好的完整代码和报告文档打包提交, 命名方式为“姓名 _ 学号.zip”。

在进行代码撰写的过程中, 为了提高计算效率, 张量和矩阵的计算请尽量使用 `torch` 提供的函数、减少 `for` 循环的使用。

2 数据准备

本次作业使用《Harry Potter》系列小说的文本作为训练语料, 已包含在 `data/harry_potter.txt` 文件中。为了将这些文本数据用以训练模型, 我们需要对原始文本进行一些预处理。

在 `prepare_data` 函数中, 原始数据被读取并储存在 `data` 中。对于每一个文章段落, 我们需要在其结尾加上特殊字符 ‘#’。你可以查看 `data/harry_potter.txt` 文件中的内容, 以确定一种快捷地添加这个特殊单词的方式。接着, 请将文本中所有的特殊字符 (如 `\n`、`\t` 等, 可以利用 `data.split()` 方法) 删除。

Task 1

补全 `dataset.py` 中的 `prepare_data` 函数, 并详细说明整个函数是如何构建词表的。

完成上述任务后运行 `dataset.py` 文件, 它会负责生成词表, 并且将文本切分为训练数据和测试数据。一个正确构建的词表应当拥有 90 个单词, 每一个单词就是一个字符。为了加深对代码的理解, 你可以现在阅读 `Vocabulary` 类 (无需在报告中解释), 我们在之后的代码补全中可能需要用到它。

接下来, 你需要完成数据集的构建和加载, 即 `dataset.py` 中的 `HarryPotterDataset` 类。在之前的作业中, 输入数据和标签是分别读取的 (比如图片和类别标签), 本次作业采用自回归预测来训练模型, 因此, 你需要基于提供的文本来分别构建输入数据和预测标签。

请按照下面的说明来完成 `HarryPotterDataset` 类:

1. 完成 `__init__` 函数。

将数据按顺序切分为 $N = \text{batch_size}$ 段等长的连续序列 (下称之为 “块”) 并储存在 `self.data` 中。注意总数据不一定能被 N 整除, 可以考虑去掉末尾的片段;

当从数据集中读取数据时, 程序会从每块中抽取一条长度为 `sequence_length` 的数据 (当未使用的数据长度不足 `sequence_length` 时, 也可以继续使用), 并组成大小为 `batch_size` 的 `batch` 来处理。请计算每个块中可以被分为多少这样的子序列, 并把其数量结果储存在 `self.sequences_in_batch` 中。

2. 完成 `__len__` 函数。

该函数的作用是返回此数据集中的总数据长度, 即被切分出的段落总数量。调用该函数之前, 数据集已完成初始化, 这意味着你可以直接使用 `__init__` 函数中的结果。

3. 完成 `__getitem__` 函数。

根据输入的 `idx` 确定返回的数据应该是哪一段 `batch`、以及这段 `batch` 中的哪一段序列, 并且返回这段序列的数据;

因为预测标签处于输入序列的后一位, 所以记得返回的数据要在 `sequence_length` 的基础上多返回一个词符。

下面是一个示例:

如果数据总共有 21 个词符 $[1, 2, \dots, 20, 21]$, `batch_size` 为 2, `sequence_length` 为 4, 那么

- 第一个 `batch` 的数据为 (`input = [[1, 2, 3, 4]; [11, 12, 13, 14]]`, `labels = [[2, 3, 4, 5]; [12, 13, 14, 15]]`)
- 第二个 `batch` 的数据为 (`input = [[5, 6, 7, 8]; [15, 16, 17, 18]]`, `labels = [[6, 7, 8, 9]; [16, 17, 18, 19]]`)
- 第三个 `batch` 的数据为 (`input = [[9]; [19]]`, `labels = [[10]; [20]]`)

如果 `idx=2`, 则会输出第二个 batch 中的第一条数据: (`input = [5, 6, 7, 8]`, `labels = [6, 7, 8, 9]`)。针对这个数据集, `self.sequences_in_batch=3`。

Task 2

请完成 `dataset.py` 中的 `HarryPotterDataset` 类。(完成代码即可, 不用在报告中写文字说明)

3 构建模型

现在我们可以开始着手构建模型, 本次作业需要完成 Transformer 模型的构建, 并且在构建过程中逐步添加部分模块, 因此, 你可以不需要完成本节的全部代码后再开始训练, 可以在完成一个代码任务后就开始训练模型 (训练过程请参考下一节的说明), 然后逐步加入新的模块。

请按照下面的说明来完成 `model.py` 中的 `HarryPotterTransformer` 类:

1. 完成 `__init__` 函数。

首先将每个词符从序号转化为词向量, 参考 `torch.nn.Embedding` (输入维度为 `vocab_size`, 词向量维度为 `feature_size`); 接着使用两层 Transformer Encoder 模块处理序列, 参考 `torch.nn.TransformerEncoderLayer` 与 `torch.nn.TransformerEncoder` (特征维度为 `feature_size`, 注意力头数量为 `num_heads`, FFN 维度为 `4 * feature_size`, dropout 设置为 0.1); 最后用一层 Linear 层作为解码层, 参考 `torch.nn.Linear` (输入维度为 `feature_size`, 输出维度为 `vocab_size`);

2. 根据模型结构完成 `forward` 函数。

前传过程中请先不使用 attention mask, 而是用 Transformer Encoder 处理全部的数据。注意 `torch.nn.TransformerEncoder` 的输入张量各个维度的顺序可能和代码其他部分不一致, 详情请查询 PyTorch 说明文档。

Task 3

请完成 `model.py` 中的 `HarryPotterTransformer` 类。(完成代码即可, 不用在报告中写文字说明)

请基于构建的网络完成训练, 绘制训练、测试的损失曲线和测试的准确率曲线。(请绘制在报告中)

意料之外, 刚刚构建的 Transformer 网络表现很差, 这是因为我们缺少了两个重要的模块: positional encoding 与 attention mask。请首先按照下面的说明完成 `PositionalEncoding` 类并将其加入到 Transformer 中:

1. 将 `pe` 初始化为 (`max_len`, `d_model`) 大小的全零矩阵, `max_len` 表示模型可接受的最大长度, `d_model` 与输入特征维度保持一致;
2. 根据下式为 `pe` 赋值:

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i + 1) = \cos(pos/10000^{2i/d_{model}}),$$

其中 `pos` 代表位置维度的索引, `i` 代表特征维度的索引;

3. 在 `HarryPotterTransformer` 类的 `__init__` 和 `forward` 函数中加入 `PositionalEncoding` 类, 注意 `PositionalEncoding` 应当在 `Embedding` 层之后, `TransformerEncoder` 层之前加入。注意每一个模块的输入输出形状。

Task 4

请完成 `model.py` 中的 `PositionalEncoding` 类并将其加入到 `HarryPotterTransformer` 类中。(完成代码即可，不用在报告中写文字说明)

请基于构建的网络完成训练，绘制训练、测试的损失曲线和测试的准确率曲线，并且分析 `positional encoding` 的作用。(请绘制在报告中)

你肯定发现了，新构建的 Transformer 网络的表现有一些奇怪，我们继续加入 `attention mask`，请按照下面的说明在 `forward` 函数中加入 `attention mask`：

1. `attention mask` 是一个 $(\text{sequence_length}, \text{sequence_length})$ 大小的上三角矩阵，其中上三角（不包含对角线）的元素都是负无穷，下三角（包含对角线）的元素都是 0；
2. 实现 `attention mask`，并将其输入到 `TransformerEncoder` 中（通过其 `mask` 接口）。

Task 5

请实现 `attention mask` 并将其加入到 `HarryPotterTransformer` 类的 `forward` 函数中。(完成代码即可，不用在报告中写文字说明)

请基于构建的网络完成训练，绘制训练、测试的损失曲线和测试的准确率曲线，并且分析 `attention mask` 的作用。(请绘制在报告中)

4 训练模型

- 准备好数据和模型后，我们可以开始训练了！训练完成之后，你可以在 `exp` 文件夹下看到训练对应的 `checkpoint`，`figure` 和 `log`。
- 注意，如果你想重新训练一个新的模型的话，记得修改 `main.py` 中的 `EXP_PATH` 参数，使新模型文件保存在另一个文件夹中（如果你不需要原来的 `checkpoint` 的话，也可以直接删除）。
- 代码会自动保存训练过程中测试成功率最高的 `checkpoint` 以及最后一个 `checkpoint`（之前的 `check-point` 则会被删除），并自动加载 `EXP_PATH` 中最新的 `checkpoint`（如果存在），所以在完整训练之前请确保该目录下没有 `checkpoint`。**请在开始实验前确认该路径。**
- 一开始你的模型可能会存在代码 `bug` 导致的训练异常，请不要等到训练全部完成之后再寻找原因修改代码，这会导致计算资源浪费和迭代效率低下。你可以通过监控模型的前期训练来快速迭代（比如将 5 epoch 的效果作为监控指标），这也是科研实践中会通常采用的做法。作为参考，我们提供不同任务第五个 epoch（代码里是 Epoch 4）的训练损失如下

	Training Loss
Task 3	2.19
Task 4	0.024
Task 5	1.44

5 文本生成

在 Task 5 中, 你应当已经得到了一个合理的输出损失和准确率, 我们现在尝试在 Task 5 的 checkpoint 的基础上构建一个 Harry Potter 小说段落生成器。你需要在 `generate.py` 的 `generate` 函数中实现如下效果:

- 将 `seed_words` 通过 `vocab` 转化为编号数列;
- 执行 `model.inference()`, 将必要的参数传递给 `model` 进行推理;
- 接受模型的输出概率分布, 并依此选择输出的字符编号: 在“greedy”策略下, 选择概率最大的那个编号、在“sampling”策略下, 按照输出概率抽取一个编号;
- 判断若输出字符不为‘#’或总输出长度未达到 `output_length`, 更新输入序列, 让模型继续预测下一个字符; 重复这个过程直到停止;
- 最终得到一个编号数组 `output_arr`, 它所指代的内容应当包含了 `seed_words` 和最后可能的停止字符‘#’。
- 在这个过程中, 请注意保证模型的输入序列不会超过模型的 `input_length`。

完成了代码补全, 你应该可以直接运行 `generate.py` 并获得一个看上去合理、但仔细读起来很奇怪的段落。记得确认 `generate.py` 中的 `EXP_PATH` 是你在 Task 5 中的实验路径。

Task 6

在默认参数 (`temperature=1`, `strategy='sampling'`) 下, 调整模型生成使用的 `seed_words`, 将你最喜欢的生成结果附在报告中。

Task 7

分别调整模型生成使用的 `temperature`、`strategy`, 观察输出段落, 你是否发现一些规律? 请简单描述不同温度 and 不同策略下的结果差异, 并简单分析原因。

最后, 祝你本次作业顺利完成。