GPU Performance

> 机器学习概论lab1
>
> **Author:**@Rosykunai
>
> **Date:** 2024年9月

这个文档是实验正文,有关实验环境配置和其它要求请查看仓库`README.md`文件

0. Intro

本次实验的目标是带你熟悉机器学习任务的基本流程,我们主要聚焦一类最经典的模型——线性模型,你将使用这类模型分别解决回归任务和分类任务,在此过程中熟悉机器学习相关工具的使用,为今后的学习打下基础.

0.1 数据集

本次实验用到的数据集分为可见和不可见两部分,可见数据集可以从Hugging Face获取,如果下载速度过慢,请使用镜像站,请认真阅读Dataset Card上的有关说明(**非常重要**!),如果不熟悉datasets库,请查阅官方文档。你将使用可见数据集完成全部的实验内容,不可见数据集作为测试集用于最终的评测。如果你使用Windows系统,默认情况下该数据集会被缓存到下面的地址:

```
C:\Users\{username}\.cache\huggingface\datasets
```

如果是Linux系统,缓存地址是用户目录下的.cache文件夹。

有了缓存后,`load_dataset()`会从缓存中读取数据集而不是重新下载。

有关数据集的概括请参阅Dataset Card, 下面是对数据集各个字段的介绍:

- `MWG`, `NWG`：每矩阵在工作组级别的二维平铺大小：{16, 32, 64, 128}(int)
- `KWG`：在工作组级别的二维平铺的内维度：{16, 32}(int)
- `MDIMC`, `NDIMC`：本地工作组大小：{8, 16, 32}(int)
- `MDIMA`, `NDIMB`：本地内存形状：{8, 16, 32}(int)
- `KWI`：内核循环展开因子：{2, 8}(int)
- `VWM`, `VWN`：每矩阵加载和存储的向量宽度：{1, 2, 4, 8}(int)
- `STRM`, `STRN`：启用单线程访问片外内存的步幅：{0, 1}(int)
- `SA`, `SB`：每矩阵手动缓存二维工作组平铺：{0, 1}(int)
- `Run_time`: 该任务的平均运行时间

0.2 文件组织

下面是对各个文件的简要介绍,更详细的内容请参考注释,

- `utils.py`: 工具函数包
- `model.py`:定义了`BaseModel`,你需要以此为基础定义自己的模型
- `trainR.py`:回归任务训练脚本
- `trainC.py`:分类任务训练脚本
- `evalR.py`:回归任务评测脚本
- `evalC.py`:分类任务评测脚本
- `submission.py`:你需要修改并提交的文件

你只需要修改`submission.py`中的内容,不得修改其它文件中的内容(它们不会被提交).

如果你在实验途中需要对数据集做一些处理或者查看当前数据集的情况,你可以使用提供的`data.ipynb`文件用于调试和分析数据集(如果你喜欢使用notebook的话),但最终需要把代码放到`submission.py`中.

1. 预测GPU的运行时间(20%)

我们使用线性回归模型来处理第一个任务,预测GPU的运行时间,即数据集的`Run_time`列。

1.1 数据预处理(8%)

在开始你的第一个机器学习任务之前,我们还得干点杂活(脏活累活),数据集的质量决定了你的算法性能上限,下面请用上你之前学过的数据预处理方法把这个数据集改造成我们需要的样子。

(a) 实现`data_preprocessing_regression()`函数,这个函数加载数据集,然后完成一个必要的预处理操作(当然你也可以添加一些其它操作,我们会保证对测试集也做同样的预处理)后返回数据集。相信大家看过数据集的介绍后都知道这个必要的操作是什么。 (4 points)

接下来的任务是划分数据集,你需要**按照8:1:1的比例**,把数据集划分为训练集，验证集和测试集(我们设置了随机种子确保每次随机划分的可复现性),在训练集上训练，根据验证集的评测结果调参；然后合并训练集和验证集训练,在测试集上评测；最后,你需要使用**所有的可见数据**训练一次,保存模型参数,作为最终的提交结果。

(b)实现`data_split_regression()`函数,这个函数把数据集划分为三个Split,然后分别加载到`Dataloader`中(请参考`utils.py`中`Dataloader`的定义),返回两个`Dataloader`:分别加载训练集和测试集。 (4 points)

- Note: 如果你的实现正确,在整个实验流程中,返回的两个`Dataloader`应该有三种情况,但是注意:不论是哪种情况,划分数据集的操作是相同的,请在划分完成后用**注释注明**train、validation和test。如果后续涉及到合并数据集的操作请使用`datasets`库或`pandas`库的相关方法,在最终提交`submission.py`时,你的`data_split_regression()`函数中应该实现的是第二种情况:合并训练集和验证集训练,在测试集上评测。

1.2 定义模型(3%)

接下来我们着手构建线性回归模型,在这里你不能直接调用`sci-kit learn`库里的相关函数(如果你感兴趣,可以在实验的最后比较你实现的model与`sklearn`库里model的性能差异)。 为了给今后的深度学习打下基础,我们参考`Pytorch`的架构实现了一套简易的框架(包含`Model`,`Loss`,`Data`,`Optimizer`),在设计上会与`Pytorch`和`sklearn`都有所区别,但在使用上会与你将来部署深度学习模型时相似。

(a)查看`model.py`中`BaseModel`的定义,然后实现`LinearRegression`。在这里,你需要把线性回归中的"权重"weight和"偏置"bias分别注册为模型的参数(Parameter)。 (2 points)

(b)实现模型的预测功能。(1 points)

1.3 定义MSELoss(3 %)

我们使用均方误差MSE作为回归问题的损失函数,你也可以选择在MSE的基础上增加一些正则化(l1、l2等)。由于我们没有实现`Pytorch`的`Tensor`(使用动态计算图自动追踪梯度),你需要再`Loss`中额外计算当前损失对各个参数的梯度用于优化模型。

(a)查看`utils.py`中`Loss`的定义,然后实现`MSELoss`的`__call__`方法计算当前的损失函数值。在这个问题中我们使用随机梯度下降法(SGD)来优化模型,你可以在`utils.py`中查看`SGD`的定义。 (1 points)

(b)实现`MSELoss`的`backward`方法,计算损失对参数的梯度。(2 points)

1.4 TrainerR(3 %)

完成了所有的准备工作！现在我们开始训练模型,要构建训练循环,你只需要把我们前面实现的小工具都用上就可以了。

(a)实现`TrainerR`的`train`方法,在train_loop中,`Dataloader`加载一个`batch`,你需要分开数据中的`feature`和`target`,让`feature`通过模型得到预测`y_pred`,再由`Loss`汇总得到均方误差和相应的梯度,最后把梯度传递给优化器更新模型参数。为了追踪训练状态,请充分使用`tqdm`的进度条功能,并记录下loss曲线. (3 points)

1.5 Train!

使用下面的命令启动默认参数配置训练脚本:

```
python trainR.py --results_path "..\results\train\"
```

你也可以把输出重定向到`log`文件中:

```
python trainR.py --results_path "..\results\train\" > trainR.log 2>&1
```

如果你使用的是Windows PowerShell,上面的重定向可能会导致报错,这时可以尝试下面的命令:

```
python trainR.py --results_path "..\results\train\" | Out-File -FilePath
train.log -Encoding utf8
```

- Note:这仍然无法完全解决问题,进度条会被输出到shell中

训练脚本还可以指定更多参数,具体细节请参考`trainR.py` 。

- Hint:这个回归问题的参数比较难调,请参考默认参数配置(不是最好的,但是能得到一个正常的结果),想要提升模型的性能请从数据质量、模型能力和超参数调整着手。调参时注意总结经验,不要盲目搜索,必要时可以参考OpenAI的相关报告。
- Note:如果你的参数比较正常,训练一轮大约需要耗费20min,再次强调调参时**不要盲目搜索**,如果你写了个脚本自动调参,请评估好训练时间和机器散热。

如果你的操作正确,训练完成后,你可以在`results/train/_Regression/`路径下找到三个文件:

- `config.yaml`: 记录这次训练的超参数配置
- `loss_list.png`: Loss曲线
- `model.pkl`:模型参数文件

1.6 评估模型性能(3%)

在验证集/测试集上评估我们刚刚训练的模型的性能,

(a)实现`eval_LinearRegression()`函数,在test_loop中,`Dataloader`加载一个`batch`，你还是需要把它分为`feature`和`target`，然后让`feature`通过模型得到预测`y_pred`,这一次,你需要统计验证集/测试集上所有预测的均值μ,当然,我们推荐你计算均方误差和相对误差$relative_error = \frac{\mu-\mu_{true}}{\mu_{true}}$来作为判断模型性能的一个参考,最终你需要返回所有预测的均值μ和相对误差。 (3 points)

使用下面的命令启动评测脚本:

```
python evalR.py --results_path "..\results\train\_Regression"
```

如果需要重定向输出,请参考1.5节。

如果你的操作正确,评测完成后,你可以在`results/train/{Date}`路径下找到文件:

- `config.yaml`: 记录当前评测的超参数

2. 对GPU的表现进行分类(20%)

在第一部分我们得到了GPU的平均运行时间,接下来请以平均运行时间为基准把数据集分为两类:

- label=1:`Run_time`>μ
- label=0:`Run_time`<μ

把`label`作为新的一列加入到数据集中,然后删掉`Run_time`列。

下面我们使用Logistic回归模型完成这个二分类任务。

- Note:第一部分的结果会对第二部分的分类效果产生较大影响,如果你认为第一部分的效果不是很好,请先完成好第一部分。

2.1 数据预处理(8%)

类似1.1节对数据集做预处理,你需要把数据集处理成用于分类的样子。

(a)实现`data_preprocessing_classification`函数,这个函数除了接受`data_path`作为参数外,还需要一个参数`mean`作为分类的基准。请传入你在第一部分预测的`Run_time`均值,**不得使用直接从数据集中计算得到的均值**,完成`label`列的构造后,别忘了删除`Run_time`列。 (4 points)

(b)划分数据集,这次我们使用完成的数据集做梯度下降,你需要返回`Tuple[Dataset]`,不需要加载`Dataloader`,划分要求同1.1-b。 (4 points)

- Note: 如果你的实现正确,在整个实验流程中,返回的两个`Dataloader`应该有三种情况,但是注意:不论是哪种情况,划分数据集的操作是相同的,请在划分完成后用**注释注明**train、validation和test。如果后续涉及到合并数据集的操作请使用`datasets`库或`pandas`库的相关方法,在最终提交`submission.py`时,你的`data_split_regression()`函数中应该实现的是第二种情况:合并训练集和验证集训练,在测试集上评测。

2.2 定义模型(3%)

接下来我们着手构建Logistic回归模型,在这里你不能直接调用`sci-kit learn`库里的相关函数(如果你感兴趣,可以在实验的最后比较你实现的model与`sklearn`库里model的性能差异)。

(a)实现`LogisticRegression`，这一次你需要把"权重"和"偏置"合并为一个参数`beta`注册为`LogisticRegression`的唯一参数(Parameter)。(2 points)

(b)实现模型的预测功能,这里要求预测当前样本的`label`为1的概率,即输出`probs`。(1 points)

2.3 定义BCELoss(3%)

我们使用二元交叉熵(BCE)作为我们二分类问题的损失函数:

$\mathcal{L}_{BCE} = -\mathbb{E}[y \log p + (1-y) \log (1-p)]$

其中y表示样本的`label`,p表示模型预测的`prob`。

(a)实现`BCELoss`的`__call__`方法,计算当前损失值,在这个问题中我们使用梯度下降法(GD)优化模型,你可以在`utils.py`中查看`GD`的定义。(1 points)

(b)实现`backward`方法,计算当前损失值对参数的梯度。(2 points)

2.4 TrainerC(3%)

下面开始训练模型。

(a)实现`TrainerC`的`train`方法,在train_loop中,这次你不需要使用`Dataloader`加载一个`batch`的数据了,你需要一次性操作整个数据集。由于我们合并了参数,别忘了给输入补充一列"1"。(3 points)

- Hint:使用GD算法通常会优化得更快,或许你可以在loss没什么变化时提前终止循环

2.5 Train!

使用下面的命令启动默认参数配置训练脚本:

```
python trainC.py --results_path "..\results\train\"
```

你也可以把输出重定向到`log`文件中:

```
python trainC.py --results_path "..\results\train\" > trainC.log 2>&1
```

如果你使用的是Windows PowerShell,上面的重定向可能会导致报错,这时可以尝试下面的命令:

```
python trainC.py --results_path "..\results\train\" | Out-File -FilePath train.log -Encoding utf8
```

- Note:这仍然无法完全解决问题,进度条会被输出到shell中

训练脚本还可以指定更多参数,具体细节请参考`trainC.py`。

- Note:如果你的参数比较正常,训练一轮大约需要耗费1min,再次强调调参时**不要盲目搜索**,如果你写了个脚本自动调参,请评估好训练时间和机器散热。

如果你的操作正确,训练完成后,你可以在`results/train/_Classification/`路径下找到三个文件:

- `config.yaml`: 记录这次训练的超参数配置
- `loss_list.png`: Loss曲线
- `model.pkl`:模型参数文件

2.6 评估模型性能(3%)

在验证集/测试集上评估我们刚刚训练的模型的性能,

(a)实现`eval_LogisticRegression`函数,这里也不需要使用`Dataloader`加载数据,我们使用准确率(ACC)作为评估分类问题的标准,别忘了在计算准确率时把`prob`转换为`label`。(3 points)

使用下面的命令启动评测脚本:

```
python evalC.py --results_path "..\results\train\_Classification"
```

如果需要重定向输出,请参考2.5节。

如果你的操作正确,评测完成后,你可以在`results/train/{Date}`路径下找到文件:

- `config.yaml`: 记录当前评测的超参数

3. [Optional]可选项

- 和`sklearn`库相比,你的模型性能如何? 为什么?
- 尝试减少训练时使用的特征数,观测模型的性能变化。

4. 回答问题(20%)

- 在对数据进行预处理时,你做了什么额外的操作,为什么?(5 points)
- 在处理分类问题时,使用交叉熵损失的优势是什么?(5 points)
- 本次实验中的回归问题参数并不好调,在实验过程中你总结了哪些调参经验?(5 points)
- 你是否做了正则化,效果如何?为什么?(5 points)

5. 反馈(1%)

引言 你可以写下任何反馈，包括但不限于以下几个方面：课堂、作业、助教工作等等。

必填 你在本次作业花费的时间大概是？(1 points)

选填 你可以随心吐槽课程不足的方面，或者给出合理的建议。若没有想法，直接忽略本小题即可。