工作区指南

自0.9.0版（或0.8.1-SNAPSHOT版）开始，ND4J提供一种额外的内存管理模式：工作区。该模式让您可以将内存重复用于循环工作负荷，而无需依赖JVM垃圾回收器来跟踪堆外内存。换言之，在工作区循环结束时，所有`INDArray`的内存内容都会被废弃。

在ND4J中使用该模式的具体方法参见这些示例。

基本的思路很简单：您可以在一个或多个工作区内部进行任何所需的操作，如果想要将一个INDArray提取出来（亦即从工作区中取出结果），只需调用`INDArray.detach()`即可获得独立的`INDArray`副本。

神经网络

作区模式可提高DL4J性能，而无需用户进行额外的配置。您只需要为具体模型的训练和推断选择资源消耗较低的模式即可。例如：

神经网络配置中的`.trainingWorkspaceMode(WorkspaceMode.SEPARATE)`和/或`.inferenceWorkspaceMode(WorkspaceMode.SINGLE)`参数。

SEPARATE（分隔式）和**SINGLE**（单一式）工作区的差别在于性能及内存占用量：

- SEPARATE模式速度稍慢，但内存使用量较少。
- SINGLE模式速度稍快，但内存使用量较多。

尽管如此，您可以在训练和推断时采用不同的模式（即用SEPARATE模式进行训练，用SINGLE模式进行推断，因为推断只需要运行一次前馈环，不涉及反向传播和更新器）。
启用工作区模式后，训练过程中使用的所有内存都可以得到重复利用及跟踪，无需JVM垃圾回收器介入。 唯一的例外是`output()`方法，如果启用工作区，该方法会在工作区内部运行前馈环，随后再将所得的`INDArray`从工作区中分离出来，这一独立的`INDArray`将会由JVM垃圾回收器来处理。

请注意：在默认情况下，训练的工作区模式设为**NONE**（不启用）。

ParallelWrapper和ParallelInference

我们也已经为`ParallelWrapper`添加了工作区模式的配置选项。如此一来，每个训练器线程都会使用一个依附于指定设备的单独工作区。

```
ParallelWrapper wrapper = new ParallelWrapper.Builder(model)
            // DataSets预提取选项。每个工作器的缓冲区大小。
            .prefetchBuffer(8)

            // 工作器数量等于GPU数量。
            .workers(2)

            // 降低平均化频率可提高性能，但有可能降低模型准确率
            .averagingFrequency(5)

            // 如设为TRUE，则每次平均化时报告模型分值
            .reportScoreAfterAveraging(false)

            // 此处有3个选项：NONE、SINGLE、SEPARATE
            .workspaceMode(WorkspaceMode.SINGLE)

            .build();
```

迭代器

我们提供异步预提取迭代器`AsyncDataSetIterator`和`AsyncMultiDataSetIterator`，两者均在内部使用。

您可以选择让这些迭代器使用一种特殊的循环工作区模式来降低内存占用量。此时工作区的大小取决于相关迭代器输出的第一个`DataSet`的内存需求，而缓冲区大小则由用户决定。如果内存需求随时间发生变化（比如用到长度可变的时间序列时），工作区也会得到调整。

注意：如果您使用的是一个自定义的迭代器或`RecordReader`，请确保第一次`next()`调用中的初始化对象不会过于庞大。请在构造器中进行这一操作，避免工作区过度扩张。

注意：使用`AsyncDataSetIterator`时，`DataSets`应在调用`next()` `DataSet`之前被使用。您不应在未调用`detach()`的情况下以任何方式存储这些`DataSets`。否则，`DataSet`中

的`INDArrays`所使用的内存最终会在`AsyncDataSetIterator`中被覆盖。

如果您由于某种原因（比如为了除错）不希望迭代器被包装成一种异步预提取方法，我们还提供特殊的包装类：`AsyncShieldDataSetIterator`和`AsyncShieldMultiDataSetIterator`。此二者都是避免预提取的简单包装类。

评估

评估通常要用到`model.output()`方法，而该方法会返回一个从工作区中分离出来的INDArray。如需在训练过程中进行定时评估，最好采用内置的评估方法。例如：

```
Evaluation eval = new Evaluation(outputNum);
ROC roceval = new ROC(outputNum);
model.doEvaluation(iteratorTest, eval, roceval);
```

上述代码将运行单个iteratorTest周期，随后更新两个（可以按需要减少/增加数量）`IEvaluation`实现，无需额外分配`INDArray`。

垃圾回收器

如果您的训练过程使用工作区，那么我们建议禁止定期调用垃圾回收器（或降低其调用频率）。具体可以采用如下代码：

```
// 这将把垃圾回收器的调用时间间隔限制为5000毫秒
Nd4j.getMemoryManager().setAutoGcWindow(5000)

// 或者可以完全禁用垃圾回收器
Nd4j.getMemoryManager().togglePeriodicGc(false);
```

请将上述代码插入`model.fit(...)`调用命令之前的某一行。

“销毁”工作区

在有些情况下，比如当RAM不足时，您可能需要释放所有并非由您控制创建的工作区；这类

情况包括评估或训练时等。

具体可采用如下代码：`Nd4j.getWorkspaceManager().destroyAllWorkspacesForCurrentThread();`

该方法会销毁调用线程之内创建的所有工作区。如果您在某些外部线程中自行创建了工作区，您可以在该线程中使用同样的方法来释放不再需要的工作区。

原文地址

https://deeplearning4j.org/workspaces

更多文档可以查看 https://github.com/sjsdfg/deeplearning4j-issues。
欢迎star