ND4J/DL4J的内存管理：工作原理

ND4J使用堆外内存存储N维数组NDArray，以便提升从本机代码（例如BLAS和CUDA库）使用NDArray时的性能。 “堆外”意味着系统分配的这部分内存位于JVM（Java虚拟机）之外，因此不受JVM垃圾回收器（GC）的管理。在Java/JVM一侧，我们只设置指向堆外内存的指针，可以传递给底层的C++代码（通过JNI），用于ND4J运算。

我们用两种方法管理内存分配：

- JVM垃圾处理器（GC）和WeakReference跟踪
- 内存工作区（MemoryWorkspaces） - 详情请参见工作区指南

尽管这两种方法之间存在差异，其背后思路是一致的：一旦（Java一侧）不再需要某些NDArray，那么与之相关联的堆外内存应当被释放，以供之后重新使用。GC和MemoryWorkspaces方法之间的区别在于内存释放的时间与方式。

- 对JVM/GC内存而言：INDArray一旦被垃圾处理器收集，其堆外内存（假设没有同时用于别处）就会被解除分配
- 对MemoryWorksaces而言：INDArray一旦离开工作区范围（例如当一个层完成正向传递/预测后），其内存就可以重新使用，无需解除分配和重新分配。 如此可以提升循环式工作任务的性能表现。

设置内存限制

就DL4J/ND4J而言，我们需要关注和设置两种类型的内存限制：堆内JVM内存上限，以及堆外内存上限。两者均由Java的命令行属性控制：

`-Xms` - 这一选项设定JVM堆在应用程序启动时使用的内存大小。
`-Xmx` - 您可以用这一选项指定JVM堆内存的限制（任何时间的上限）。内存分配量（JVM可用）最多不超过此处设置的值。
`-Dorg.bytedeco.javacpp.maxbytes` - 您可以用这一选项指定堆外内存上限。
`-Dorg.bytedeco.javacpp.maxPhysicalBytes` - 同样用于堆外内存，这一选项通常应当设为与`maxbytes`相等

示例：设置1GB初始堆内内存、2GB堆内内存上限、8GB堆外内存：

```
-Xms1G -Xmx2G -Dorg.bytedeco.javacpp.maxbytes=8G -Dorg.bytedeco.javacpp.maxPhysicalBytes=8G
```

问题：有几件事值得注意

- 对于GPU系统，由于堆外存储器（通过NDArrays）映射到GPU，所以maxbytes和maxphysicalbytes设置当前也有效地定义了GPU的内存限制 - 请在下面的GPU部分中阅读更多关于此内容的内容。
- 对于许多应用程序来说，JVM堆使用的RAM应当较少，而将更多RAM用于堆外，因为NDArray都存储在堆外。如果分配过多内存给JVM堆，就没有足够的内存留给堆外使用了。
- 如果未指定JVM堆的内存上限，JVM默认将使用系统RAM总量的1/4。
- 如果未指定堆外内存上限，默认将使用JVM堆内存上限（Xmx）的2倍。例如，`-Xmx8G`表明JVM堆最多可以使用8GB，而ND4J默认在堆外可以使用16GB。
- 在内存有限的环境中，不宜将`-Xmx`值设得过高，同时使用`-Xms`选项。理由同前：防止留给堆外的内存量不足。以一个16GB内存的系统为例。假设您设置-Xms14G，这意味着16GB的内存中有14GB将被分配给JVM，只给堆外内存留下2GB（还要包括操作系统和其他所有程序）。

内存映射文件

自0.9.2-SNAPSHOT版本起，使用`nd4j-native`后端时可以用内存映射文件代替RAM。内存映射文件虽然速度比RAM慢，但可以用特殊的方式分配内存。
代码示例如下：

```
WorkspaceConfiguration mmap = WorkspaceConfiguration.builder()
                .initialSize(1000000000)
                .policyLocation(LocationPolicy.MMAP)
                .build();

try (MemoryWorkspace ws = Nd4j.getWorkspaceManager().getAndActivateWorkspace(mmap,
"M2")) {
    INDArray x = Nd4j.create(10000);
}
```

此处将有1GB的临时文件被创建并映射至内存，而NDArray x将在这一空间中创建。 很显然，这一选项最适合在需要RAM无法容纳的NDArray时使用。

GPU

使用GPU时，一种典型的情况是CPU的RAM大于GPU RAM。

如果GPU RAM小于CPU RAM，我们需要监控有多少RAM被用于堆外。您可以通过上文所述的JavaCPP选项检查这一点。

此处需要注意的是，用于分配的GPU内存量等于您所指定的堆外内存量。我们使用的GPU内存量不会超过这一上限。您也可以（但不推荐这样做）指定超过GPU内存量的堆空间，但这样一来GPU在运行任务时就会出现RAM不足的情况。我们同样也把CPU的RAM分配给堆外内存。其目的是为了确保CPU到GPU的通信效率较高，让CPU能访问NDArray中的数据，而无需从GPU中抓取数据。

如果JavaCPP或者您的GPU出现内存不够的错误，甚至计算速度降低时（GPU内存有限导致），那么您可能需要减少批次大小，或者可以增加JavaCPP允许分配的堆外内存量。

请尝试将堆外内存量设为与GPU的RAM相同。同时请记得把堆空间设置得小一些。

请注意，如果您的GPU RAM小于2G，那么这个GPU可能不适合用于深度学习。此时应考虑使用您的CPU。

典型的深度学习工作负荷最少应有4G的RAM。但我们不建议只配备4G的RAM，如果要运行深度学习任务，GPU至少应该有8G的RAM。

原文地址

https://deeplearning4j.org/memory

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