Deeplearning4j - Arbiter 概述

- 原文地址：https://deeplearning4j.org/docs/latest/arbiter-overview
- 翻译者原文地址：http://bewithme.iteye.com/blog/2431677
- 群友 @风一样的男子 提供

超参数优化

机器学习有一个参数集合，必须在任何训练开始之前选择。 这些参数就是所谓 的超参数。 一些超参数的例子是 K 邻近值算法的“K”和支持向量机中的正则化参数。 神经网络，比较特别，有很多的超参数。它们中的一些定义了神经网络的结构，像层的数量和它们的大小。其它一些定义了学习过程比如说学习率和正则化。

传统上，这些选择是根据现有的经验法则做出的，或者经过大量的试验和错误后做出的，这两者都不太理想。 无疑的这些参数选择会在学习取得的结果上有重要影响。 超参数优化尝试使用应用搜索策略的软件自动化该过程。

Arbiter（阿比特）

Arbiter 是企业机器学习 / 深度学习工具的 dl4j 套件之一. 它专用于 dl4j 神经网络的创建和导入时的参数优化 . 它允许用户为超参数设置搜索空间，并运行网 格搜索或随机搜索来在基于给定的评分指标上选择最好的配置. 什么时候用 Arbiter?

Arbiter 可以用作寻找表现好的模型，潜在的为你节约调优模型超参数的时间，但是牺牲了更大的计算时间. 注意 Arbiter 不会完全自动化神经网络调优的过程，用户仍需要指定搜索空间. 这个搜索空间定义了每个超参数可用的值的范围（例如：学习率允许的最大值和最小值）. 如果这个搜索空间选择过小， Arbiter 可能找不到任何好的模型。

添加以下到你的 pom.xml 来在你的工程中引用 Arbiter，${arbiter.version} 是最新的发布版本。

```
<!-- Arbiter - used for hyperparameter optimization (grid/random search) -->
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>arbiter-deeplearning4j</artifactId>
    <version>1.0.0-beta2</version>
</dependency>
<dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>arbiter-ui_2.11</artifactId>
    <version>1.0.0-beta2</version>
</dependency>
```

阿比特还提供方便的用户界面来帮助把优化的结果可视化。使用阿比特的先决条件是用户应熟悉 dl4j 中的神经网络配置，多层神经网络配置和计算图配配置类。

使用

本节将概述使用 Arbiter 所必需的重要组成。接下来的章节将深入细节。在最高级别，设置超参数优化涉及设置一个优化配置和通过 IOptimizationRunner 来运行它。如下是一些演示在优化配置中建造者模式的代码：

```
OptimizationConfiguration configuration = new OptimizationConfiguration.Builder()
    .candidateGenerator(candidateGenerator)
    .dataSource(dataSourceClass,dataSourceProperties)
    .modelSaver(modelSaver)
    .scoreFunction(scoreFunction)
    .terminationConditions(terminationConditions)
    .build();
```

如上所述，设置优化配置需要:

1. 候选生成器: 为评估提出候选 (i.e., 超参数配置) . 候选基于一些策略创建. 目前支持随机搜索和网格搜索.有效的候选配置取决于与候选生成器相关连的超参数空间.
2. 数据源: 数据源在后台使用，用于为生成的候选提供训练和测试数据。
3. 模型保存器: 指定每个超参数优化器运行的结果将如何保存. 例如，是否保存到本地磁盘，数据库，HDFS 系统，或只是简单的保存在内存. 评分函数: 一个单数字度量值，用于寻找最小化或最大化来决定最好的候选 . 如模型损失或分类精确度。
4. 终止条件: 决定超参数优化在什么时候停止.如一定数量的候选已被评估，已过了一定量的计算时间.然后将优化配置连同任务创建者一起传递给优化运行程序。

如果创建的候选是多层网络，这个设置如下:

```
IOptimizationRunner runner = new LocalOptimizationRunner(configuration, new
MultiLayerNetworkTaskCreator());
```

作为可选项，如果创建的候选为计算图，这个设置如下:

```
IOptimizationRunner runner = new LocalOptimizationRunner(configuration, new
ComputationGraphTaskCreator());
```

目前为止运行器惟一可用的选项为 LocalOptimizationRunner，用于在单台机器上执行学习（在当前 JAVA 虚拟机上）.原则上，其它执行方法（如，在 spark 或云计算机器）可以被实现.
这里总结了建立超参数优化运行的步骤：

1. 指定超参数的搜索空间

2. 为超参数搜索空间指定一个候选生成器
3. 接下来的步骤可以按任意的顺序执行:
4. 指定一个数据源
5. 指定一个模型保存器
6. 指定一个评分函数
7. 指定终止条件
8. 以下步骤必须按顺序执行:
9. 使用如上的 2-6 来构建一个优化配置
10. 和优化运行器一起运行.

超参数的搜索空间

Arbiter 的 ParameterSpace 类定义了一个给定的超参数可能使用可接受的值的范围。参数空间可以是一个简单的，如参数空间定义一个双精度值的连续范围（学习率） 或是复杂的，如多个嵌套参数空间，与多层空间（为 MultilayerConfiguration 定义搜索空间）情况类似。

多层空间和计算图空间

多层空间和计算图空间是 Arbiter 对于 dl4j 的多层配置和计算图配置的副本。它们用于在多层配置和计算图配置中为可用的超参数设置超参数空间。除此之外，这些用户也可以设置训练次数或是一个早停配置来指示在每个候选神经网络训练时什么候应该停止。如果一个早停配置和训练次数都被指定，早停会优先使用。

如果用户熟悉整数，连续和离散参数空间，和层空间和更新器空间，那么设置多层空间和计算图空间是非常容易直接的。惟一需要注意的警告是虽然设置权重约束是可能的，但是作为神经网络配置一部份的 l1Bias 和 l2Bias 必须在多层空间中的每层/层空间基础上设置。通常所有可通过建造器使用的属性 / 超参数将采用固定值或该类型的参数空间。这意味着几乎多层配置的每一个层面都会被扫描用于对各种结构和初始化值进行彻底测试。

这里是一个多层空间的简单例子：

```
ParameterSpace<Boolean> biasSpace = new DiscreteParameterSpace<>(new Boolean[]{true, f
alse});
ParameterSpace<Integer> firstLayerSize = new IntegerParameterSpace(10,30);
ParameterSpace<Integer> secondLayerSize = new MathOp<>(firstLayerSize, Op.MUL, 3);
ParameterSpace<Double> firstLayerLR = new ContinuousParameterSpace(0.01, 0.1);
ParameterSpace<Double> secondLayerLR = new MathOp<>(firstLayerLR, Op.ADD, 0.2);

MultiLayerSpace mls =
    new MultiLayerSpace.Builder().seed(12345)
            .hasBias(biasSpace)
            .layer(new DenseLayerSpace.Builder().nOut(firstLayerSize)
                    .updater(new AdamSpace(firstLayerLR))
```

```
                        .build())
                .layer(new OutputLayerSpace.Builder().nOut(secondLayerSize)
                        .updater(new AdamSpace(secondLayerLR))
                        .build())
                .setInputType(InputType.feedForward(10))
    .numEpochs(20).build(); //Data will be fit for a fixed number of epochs
```

需要特别注意的是 Arbiter 在多层空间中改变层数量的能力。下面是一个简单的示例，演示了如何为加权损失函数设置参数搜索空间：

```
ILossFunction[] weightedLossFns = new ILossFunction[]{
    new LossMCXENT(Nd4j.create(new double[]{1, 0.1})),
        new LossMCXENT(Nd4j.create(new double[]{1, 0.05})),
            new LossMCXENT(Nd4j.create(new double[]{1, 0.01}))};

DiscreteParameterSpace<ILossFunction> weightLossFn = new DiscreteParameterSpace<>(weightedLossFns);
MultiLayerSpace mls =
    new MultiLayerSpace.Builder().seed(12345)
        .addLayer(new DenseLayerSpace.Builder().nIn(10).nOut(10).build(),
            new IntegerParameterSpace(2, 5)) //2 to 5 identical layers
        .addLayer(new OutputLayerSpace.Builder()
            .iLossFunction(weightLossFn)
            .nIn(10).nOut(2).build())
        .backprop(true).pretrain(false).build();
```

上面创建的两到五层将是相同的（堆叠）。目前为止 Arbiter 不支持创建独立层，最后，也可以创建固定数量的相同层，如下面的示例所示：

```
DiscreteParameterSpace<Activation> activationSpace = new DiscreteParameterSpace(new Activation[]{Activation.IDENTITY, Activation.ELU, Activation.RELU});
MultiLayerSpace mls = new MultiLayerSpace.Builder().updater(new Sgd(0.005))
    .addLayer(new DenseLayerSpace.Builder().activation(activationSpace).nIn(10).nOut(10).build(),
        new FixedValue<Integer>(3))
    .addLayer(new OutputLayerSpace.Builder().iLossFunction(new LossMCXENT()).nIn(10).nOut(2).build())
    .backprop(true).build();
```

在这个例子中，网格搜索将创建三个独立的体系结构。它们将在每一个方面都相同，但是在非输出层中被选择的激活函数中是不同的。另外，要注意的是，在每个体系结构中创建的层是相同的（堆叠）。创建计算图形空间与多层空间非常相似。然而，目前只支持固定的图形结构。下面是一个简单的例子，演示了设置计算图空间：

```
ComputationGraphSpace cgs = new ComputationGraphSpace.Builder()
                .updater(new SgdSpace(new ContinuousParameterSpace(0.0001, 0.1)))
                .l2(new ContinuousParameterSpace(0.2, 0.5))
                .addInputs("in")
                .addLayer("0",new  DenseLayerSpace.Builder().nIn(10).nOut(10).activatio
```

```
n(
                new DiscreteParameterSpace<>(Activation.RELU,Activation.TANH).build(),"in"
)

            .addLayer("1", new OutputLayerSpace.Builder().nIn(10).nOut(10)
                                    .activation(Activation.SOFTMAX).build(), "0")
            .setOutputs("1").setInputTypes(InputType.feedForward(10)).build();
```

JSON 序列化

多层空间，计算图空间和优化配置有 toJson 方法也有 fromJson 方法。你可以存储 JSON 表示以供进一步使用。

指定一个前面提到过的候选生成器，阿比特当前支持网格搜索和随机搜索。

设置一个随机搜索是很简单的，如下所示：`MultiLayerSpace mls; … CandidateGenerator candidateGenerator = new RandomSearchGenerator(mls);`

设置一个网格搜索同样简单。通过网格搜索，用户还可以指定离散化计数和模式。离散化计数决定连续参数有多少值被包含在其中。例如，一个在[0,1] 区间的连续参数，在离散化计数为 3 时被转化为[0.0,0.5,1.0] 该模式决定了生成候选的方式。候选可以按顺序创建或随机创建。按照顺序，第一超参数变化最快，因此最后一个超参数变化最慢。请注意，这两种模式将导致在相同的一组候选，只是顺序不同。

这里是一个网格搜索如何用一个离散化计数为 4 的按顺序被设置一个例子。

```
CandidateGenerator candidateGenerator = new GridSearchCandidateGenerator(mls, 4,
 GridSearchCandidateGenerator.Mode.Sequential);
```

指定数据源

数据源接口定义了用于训练不同候选的数据来源。实现起来非常简单。注意到需要定义一个没有参数的构造器。取决于用户的需要，数据源可以通过配置实现，就像小批理的大小。一个使用 MNIST 数据集的数据源的简单实现在 repo 例子中可以使用，它将在后面的指引中涉及。需要注意的是很重要的是训练次数（早停配置也是一样）可以通过多层空间和计算图构建器被设置。

指定一个模型 / 结果 保存器

Arbiter 目前支持内存保存到磁盘（文件模型保存器）来保存模型或在内存存储结果（内存结果保存器）
对于大模型来说内存结果保存器很明显是不推荐的。设置它们是很简单的。文件模型保存器构造器使用一个字符

路径。它保存配置，参数和分数到：baseDir/0/, baseDir/1/, 等，它们的索引由 OptimizationResult.getIndex() 方法提供，内存结果保存器不需要参数。
指定一个评分函数这里有三个主要的类用于评分函数：EvaluationScoreFunction，ROCScoreFunction 和 RegressionScoreFunction。
EvaluationScoreFunction 使用一个 dl4j 评估指标，可用的指标是 ACCURACY, F1, PRECISION, RECALL, GMEASURE, MCC。这里有一个使用 accuracy 的简单例子: ScoreFunction scoreFunction = new EvaluationScoreFunction(Evaluation.Metric.ACCURACY);
ROCScoreFunction 在测试数据集上计算 AUC(曲线面积) 或 AUPRC（精度/召回曲线面积）。支持不同的曲线面积类型（ROC, ROCBinary 和 ROCMultiClass）。这里是一个简单的使用 AUC 的例子：ScoreFunction sf = new ROCScoreFunction(ROCScoreFunction.ROCType.BINARY, ROCScoreFunction.Metric.AUC));
RegressionScoreFunction 用于回归和支持所有 dl4j 的回归指标（MSE, MAE, RMSE, RSE, PC, R2）这里是一个简单的例子：ScoreFunction sf = new RegressionScoreFunction(RegressionEvaluation.Metric.MSE);

指定一个终止条件

Arbiter 目前仅支持两种终止条件-最大时间条件最大候选条件。最大时间条件是为超参数优化指定一个停止时间。最大候选条件为超参数优化停止前指定一个候选的最大数量。终止条件可以指定为一个列表。如果任何一个条件满足超参数优化将停止。这里是一个简单的例子，运行将会在 15 分中后停止或在训练了 10 个候选之后，取决于哪个条件先到。

```
TerminationCondition[] terminationConditions = {
    new MaxTimeCondition(15, TimeUnit.MINUTES),
    new MaxCandidatesCondition(10)
};
```

运行在 MNIST 数据集的例子

dl4j 的例子中包括一个在 MNIST 数据集上的 BasicHyperparameterOptimizationExample 例子。用户可以在这里浏览这个简单的例子。这个例子也通过设置阿比特的界面来实现。阿比特使用与 dl4j 界面相同的存储和持久化方法。更多的有关界面的文档可以在这里找到。界面可以通过 http://localhost:9000/arbiter 访问。

超参数调优提示

请参阅斯坦福大学 CS241N 课程的超参数优化的优秀章节。这些技巧的总结如下：

- 在网格搜索和随机搜索中优先使用随机搜索。要比较随机和网格搜索方法，请参阅用于超参数优化的随机搜索（Bergstra 和 Bengio，2012）。

- 从粗略到精细运行搜索（以粗略的参数开始搜索一个或两个训练次数，挑最好的候选来做精细搜索，用更多的训练次数，迭代 ）
- 为某些超参数（如学习率、L2 等）使用对数均匀分布
- 注意接近参数搜索空间边界的值。