Dl4j学习率衰减策略

```
package org.deeplearning4j.nn.conf;

/**
 * Learning Rate Policy
 *
 * How to decay learning rate during training.
 *
 * <p><b>None</b> = do not apply decay policy aka fixed in Caffe <br>
 * <p><b>Exponential</b> = applies decay rate to the power of the # batches  <br>
 * <p><b>Inverse</b> = divide learning rate by negative (1 + decay rate * # batches)^power <br>
 * <p><b>Poly</b> = polynomial decay that hits 0 when iterations are complete <br>
 * <p><b>Sigmoid</b> = sigmoid decay rate <br>
 * <p><b>Step</b> = decay rate to the power of the floor (nearest integer) of # of batches by # of steps <br>
 * <p><b>Schedule</b> = rate to use at a specific iteration <br>
 * <p><b>Score</b> = apply decay when score stops improving <br>
 */

// TODO provide options using epochs in addition to iterations

public enum LearningRatePolicy {
    None, Exponential, Inverse, Poly, Sigmoid, Step, TorchStep, Schedule, Score
}
```

---

dl4j的学习率衰减策略应用部分是在反向传播计算完地图之后，调用Updater.update()方法对梯度进行更新并且进行梯度的衰减。

调用学习率衰减的为`package org.deeplearning4j.optimize.solvers`包下的`BaseOptimizer`抽象类中的`updateGradientAccordingToParams(Gradient gradient, Model model, int batchSize)`方法中的`updater.update(layer, gradient, getIterationCount(model), batchSize);`语句。

- 后面衰减策略所使用的iteration为Model的总迭代次数
- 衰减率由`NeuralNetConfiguration.Builder()`的`lrPolicyDecayRate()`方法进行配置，衰减率通常为0~1中间的一个值。

Exponential

```
newLr = lr * Math.pow(decayRate, iteration);
```

$$newLr = lr \times decayRate^{iteration}$$

Inverse

```
newLr = lr / Math.pow((1 + decayRate * iteration), conf.getLrPolicyPower());
```

$$newLr = \frac{lr}{1 + (decayRate \times iteration)^{conf.getLrPolicyPower()}}$$

注： LrPolicyPower由用户自定义设置。默认值为0。

Step

```
newLr = lr * Math.pow(decayRate, Math.floor(iteration / conf.getLrPolicySteps()));
```

$$newLr = lr \times decayRate^{\left\lfloor \frac{iteration}{conf.getLrPolicySteps()} \right\rfloor}$$

注： lrPolicySteps 由用户自定义，默认值为0。

TorchStep

```
//当模型的总迭代次数>1次且设置的lrPolicySteps 正好是当前总迭代次数的倍数的时候调用
if (iteration > 1 && conf.getLrPolicySteps() % iteration == 0) {
```

```
        newLr = lr * decayRate;
    } else {
        newLr = lr;
    }
```

$$newLr = lr \times decayRate$$

Poly

```
newLr = lr * Math.pow((1 - ((double) iteration) / conf.getNumIterations()), conf.getLrPolicyPower());
```

$$newLr = lr * (1 - \frac{iteration}{conf.getNumIterations()})^{conf.getLrPolicyPower()}$$

注： conf.getNumIterations()，是对于每一个miniBatch的数据中的迭代次数。

Sigmoid

```
newLr = lr / (1 + Math.exp(-decayRate * (iteration - conf.getLrPolicySteps())));
```

$$newLr = \frac{lr}{1 + e^{-decayRate \times (iteration - conf.getLrPolicySteps())}}$$

Schedule

```
if (baseLayer.getLearningRateSchedule().containsKey(iteration)) {
    newLr = baseLayer.getLearningRateSchedule().get(iteration);
} else {
    newLr = lr;
}
```

根据前面设置的Schedule来更改对应的学习率，例如设置的Schedule为：

```
// Map为<iterations, learningRate>的key-value对
```

```
Map<Integer, Double> schedule = new HashMap<>();
schedule.put(200, 0.01);
schedule.put(60000, 0.001);
schedule.put(80000, 0.0001);
```

就是在模型迭代第200次的时候，学习率变更为0.01；在第60000次的时候变更为0.001...依次类推

Score

Score的触发条件和前面几个都并不一样，触发的代码为：

```
@Override
public boolean checkTerminalConditions(INDArray gradient, double oldScore, double score, int i) {
    for (TerminationCondition condition : terminationConditions) {
        //log.info("terminations: {}", condition);
        if (condition.terminate(score, oldScore, new Object[] {gradient})) {
            log.debug("Hit termination condition on iteration {}: score={}, oldScore={}, condition={}", i, score,
                            oldScore, condition);

            //触发EpsTermination，且当前网络层不为空且衰减策略为Score的时候调用
            if (condition instanceof EpsTermination && conf.getLayer() != null
                            && conf.getLearningRatePolicy() == LearningRatePolicy.Score) {
                model.applyLearningRateScoreDecay();
            }
            return true;
        }
    }
    return false;
}
```

其中策略的实现方式基本为：

```
@Override
public void applyLearningRateScoreDecay() {
    for (Map.Entry<String, Double> lrPair : conf.getLearningRateByParam()
```

```
.entrySet())
            conf.setLearningRateByParam(lrPair.getKey(),
                                    lrPair.getValue() * (conf.getLrPolicyDecayRate() + Nd4j.EPS_THRESHOLD));
}
```

$$newLr = lr \times (conf.getLrPolicyDecayRate() + Nd4j.EPS\_THRESHOLD)$$

注：

1. Nd4j.EPS_THRESHOLD的默认值为 $1e^{-5}$
2. 使用这个策略的时候，网络模型损失函数得分容易进入`ZeroDirectionTermination`使得模型学习率不再衰减，使得模型停止更新。

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