基于Spark的极限学习机算法并行化研究

刘 鹏[1,2] 王学奎[1,3] 黄宜华[4] 孟 磊[1,2] 丁恩杰[1,2,3]

(中国矿业大学物联网(感知矿山)研究中心 徐州 221008)[1]

(矿山互联网应用技术国家地方联合工程实验室 徐州 221008)[2]

(中国矿业大学信息与控制工程学院 徐州 221116)[3]

(南京大学计算机系 PASA 大数据实验室 南京 210023)[4]

摘 要 极限学习机算法虽然训练速度较快,但包含了大量矩阵运算,因此其在面对大数据量时,处理效率依然缓慢。在充分研究Spark分布式数据集并行计算机制的基础上,设计了核心环节矩阵乘法的并行计算方案,并对基于Spark的极限学习机并行化算法进行了设计与实现。为方便性能比较,同时实现了基于Hadoop MapReduce的极限学习机并行化算法。实验结果表明,基于Spark的极限学习机并行化算法相比于Hadoop MapReduce版本的运行时间明显缩短,而且若处理数据量越大,Spark在效率方面的优势就越明显。

关键词 极限学习机,并行化,Spark,RDD,Hadoop,MapReduce

中图法分类号 TP18 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.006

Study of ELM Algorithm Parallelization Based on Spark

LIU Peng[1,2] WANG Xue-kui[1,3] HUANG Yi-hua[4] MENG Lei[1,2] DING En-jie[1,2,3]

(Internet of Things Perception Mine Research Centre, China University of Mining and Technology, Xuzhou 221008, China)[1]

(National and Local Joint Engineering Laboratory of Internet Application Technology on Mine, Xuzhou 221008, China)[2]

(School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China)[3]

(PASA Big-data Laboratory, Department of Computer Science, Nanjing University, Nanjing 210023, China)[4]

Abstract Extreme learning mechine(ELM) has high training speed, but with lots of matrix operations, it remains poor efficiency while applied to massive amount of data. After thorough research on parallel computation of Spark resilient distributed dataset (RDD), we proposed and implemented a parallelized algorithm of ELM based on Spark. And for convenience of performance comparison, Hadoop-MapReduce-based version was also implemented. Experimental results show that the training efficiency of the Spark-based ELM parallelization algorithm is significantly improved than the Hadoop-MapReduce-based version. If the amount of data processed is greater, the advantage of Spark in efficiency is more obvious.

Keywords ELM, Parallelization, Spark, RDD, Hadoop, MapReduce

1 引言

为解决传统神经网络算法训练速度缓慢的问题,2006年黄广斌提出了极限学习机算法ELM(Extremely Learning Machine)[1]。大量研究表明,ELM算法总体上明显提高了训练速度,在很多情况下可以提高成百上千倍,而且在速度提高的同时,准确率及泛化性能也有不同程度的提高[1-3]。

由于ELM中被处理的数据需要事先加载至内存,以及算法中有大量的矩阵运算,因此在面对大规模训练数据时,ELM的处理效率急剧下降,面临着巨大挑战。2013年中科院何清[4]首次提出了基于MapReduce[5-6]设计〈key,value〉键值对来处理ELM并行计算问题。2014年浙江大学陈娇燕[7]将ELM分成多个子模块,并基于MapReduce集群对分布式数据块并行训练。但上述工作的核心即并行化工作都研究得不够深入,程序运行效率依然有待提高。

Spark[8-9]是新一代基于内存计算的大数据处理平台,相比于Hadoop MapReduce做出了大量重要的改进。譬如,Hadoop MapReduce的每个作业执行结果都需要写入磁盘HDFS,因此在包含多作业的复杂任务执行中需不断地读写磁盘,耗费了大量时间;而Spark创新性地引入了内存分布数

到稿日期:2016-10-11 返修日期:2016-11-12 本文受国家重点研发计划:矿山安全生产物联网关键技术与装备研发(2017YFC0804400,2017YFC0804401),国家自然科学基金项目(61471361,41302203)资助。

刘 鹏(1973-),男,博士,副教授,CCF会员,主要研究领域为大数据并行处理技术及其在矿山物联网中的应用,E-mail:liupeng@cumt.edu.cn;王学奎(1993-),男,硕士生,主要研究领域为并行化计算、深度学习,E-mail:15105218783@163.com;丁恩杰(1962-),男,博士,教授,主要研究领域为矿山云服务及矿山大数据处理技术等,E-mail:enjied@cumt.edu.cn(通信作者)。

据集 RDD(Resilient Distributed Dataset),其中间计算结果都可以保存在内存之中,只将最后结果输出至磁盘,因此可以显著加速作业的执行。文献[10]表明,与基于 Hadoop 的实现相比,基于 SparkR 的并行分类算法的实现的处理效率明显提高。

在对 Spark RDD 并行计算机制进行深入研究的基础上,本文的主要贡献在于:提出了基于 Spark 的 ELM 并行化算法,尤其在其中的矩阵运算并行化部分做出了一些创新性的工作,并通过与基于 Hadoop MapReduce 的 ELM 并行算法作比较,体现出本算法在运行效率方面的优越性。

本文第 2 节对 ELM 串行算法进行简单介绍;第 3 节分别详细介绍基于 Hadoop 和 Spark 的并行 ELM 算法的设计及实现;第 4 节通过实验对基于 Spark 以及 Hadoop 的 ELM 并行算法进行性能比较并得出结论;最后进行总结和展望。

2 ELM 串行算法

相比于传统神经网络算法,ELM 是一个结构简单的单隐层前馈神经网络,其主要设计目标是显著提高模型的训练速度。经典 ELM 串行算法的步骤如下[1]。

ELM 算法:假设单隐层前馈神经网络有 $\widetilde{N}$ 个隐层节点,激活函数为 $g(x)$,训练样本集 $\aleph=\{(\boldsymbol{x}_i,\boldsymbol{t}_i)\mid \boldsymbol{x}_i\in\boldsymbol{R}^n,\boldsymbol{t}_i\in\boldsymbol{R}^m,i=1,\cdots,\widetilde{N}\}$。

第一步 随机确定输入权重向量 $\boldsymbol{\omega}_i$ 以及隐层偏差 b_i,$i=1,\cdots,\widetilde{N}$。

第二步 计算单隐层前馈神经网络输出矩阵 $\boldsymbol{H}$,如式(1)所示:

$$\boldsymbol{H}(\boldsymbol{\omega}_1,\cdots,\boldsymbol{\omega}_{\widetilde{N}},b_1,\cdots,b_{\widetilde{N}},\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N)=\begin{bmatrix} g(\boldsymbol{\omega}_1\cdot\boldsymbol{x}_1+b_1) & \cdots & g(\boldsymbol{\omega}_{\widetilde{N}}\boldsymbol{x}_1+b_{\widetilde{N}}) \\ \vdots & \vdots & \vdots \\ g(\boldsymbol{\omega}_1\cdot\boldsymbol{x}_N+b_1) & \cdots & g(\boldsymbol{\omega}_{\widetilde{N}}\boldsymbol{x}_N+b_{\widetilde{N}}) \end{bmatrix}_{N\times\widetilde{N}} \tag{1}$$

第三步 计算输出权重矩阵 β 的唯一最优解:

$$\hat{\beta}=\boldsymbol{G}\boldsymbol{T} \tag{2}$$

其中,$\boldsymbol{T}=[t_1,\cdots,t_N]^{\mathrm{T}}$,$G$ 为单隐层前馈神经网络输出矩阵 $\boldsymbol{H}$ 的广义逆矩阵。求解广义逆矩阵的方法包括正交化、正交投影、迭代法及奇异值分解等。在 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 非奇异(满秩)的情况下,正交投影可以用来求解广义逆矩阵,并且有 $G=(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})^{-1}\boldsymbol{H}^{\mathrm{T}}$,这在 Ferrari 的经典文献中有应用[11]。因此可通过:

$$\hat{\beta}=(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})^{-1}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T} \tag{3}$$

求解输出权重矩阵 β 的唯一最优解 $\hat{\beta}$。

3 ELM 并行化算法

根据 $\hat{\beta}=(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})^{-1}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T}$ 可知,在求解输出权重矩阵唯一最优解的过程中,主要包含 3 类矩阵运算,分别是生成单隐层前馈神经网络输出矩阵 $\boldsymbol{H}$、矩阵 $\boldsymbol{H}$ 求逆运算及矩阵乘法运算。

3.1 基于 Hadoop 的矩阵乘法并行化

假设有两个矩阵 $\boldsymbol{A}=\begin{bmatrix}1&2&3\\4&5&0\\7&8&9\\19&11&12\end{bmatrix}$,$\boldsymbol{B}=\begin{bmatrix}10&15\\0&2\\11&9\end{bmatrix}$,则 $\boldsymbol{C}=\boldsymbol{A}\boldsymbol{B}=\begin{bmatrix}43&46\\40&70\\169&202\\232&280\end{bmatrix}$,可以直接通过单机计算得到这样的两个小矩阵相乘的结果。但是若采用同样的单机直接相乘策略,在大数据背景下的大矩阵相乘则会带来难以忍受的长时间消耗。基于 Hadoop MapReduce,将大矩阵相乘按行列位置分解为多个独立的带有标记的元素,并两两相乘进行并行化,以此核心思想来设计矩阵乘法的并行化,具体方案如下:根据矩阵相乘的原则,两个矩阵只有在前者的列数和后者的行数相等时才能够相乘,而且相乘所得结果的行列数与相乘的两个矩阵中的元素坐标存在密切关系,根据这种关系可以将矩阵的各个元素切分开并赋予每个元素一个关键字,以便在并行化的过程中能够非常明确地知道一个元素应该与哪些元素相乘,以及哪些乘积结果可以相加从而得到最终所求元素。矩阵乘并行化方案如图 1—图 4 所示。

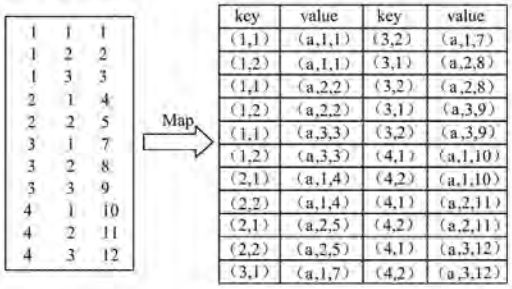

图 1 矩阵 A 的 Map 过程

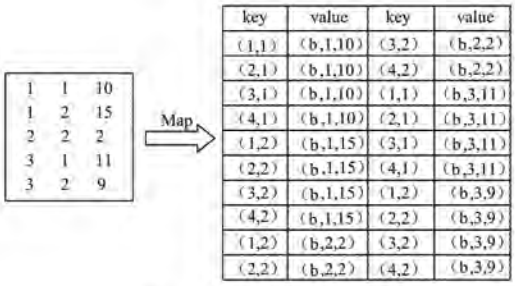

图 2 矩阵 B 的 Map 过程

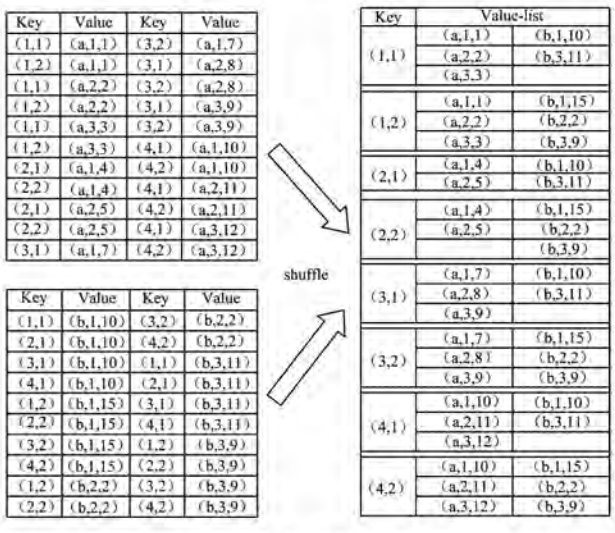

图 3 矩阵乘 $A\times B$ 的 shuffle 过程

Key	Value-list	
(1,1)	(a,1,1)	(b,1,10)
	(a,2,2)	(b,3,11)
	(a,3,3)	
(1,2)	(a,1,1)	(b,1,15)
	(a,2,2)	(b,2,2)
	(a,3,3)	(b,3,9)
(2,1)	(a,1,4)	(b,1,10)
	(a,2,5)	(b,3,11)
(2,2)	(a,1,4)	(b,1,15)
	(a,2,5)	(b,2,2)
		(b,3,9)
(3,1)	(a,1,7)	(b,1,10)
	(a,2,8)	(b,3,11)
	(a,3,9)	
(3,2)	(a,1,7)	(b,1,15)
	(a,2,8)	(b,2,2)
	(a,3,9)	(b,3,9)
(4,1)	(a,1,10)	(b,1,10)
	(a,2,11)	(b,3,11)
	(a,3,12)	
(4,2)	(a,1,10)	(b,1,15)
	(a,2,11)	(b,2,2)
	(a,3,12)	(b,3,9)

Reduce

Key	Value
(1,1)	43
(1,2)	46
(2,1)	40
(2,2)	70
(3,1)	169
(3,2)	202
(4,1)	232
(4,2)	280

图 4　矩阵乘 $A\times B$ 的 Reduce 过程

在图 1 和图 2 中，分别对矩阵 A 和矩阵 B 做出不同的 Map 操作，然后将 Map 函数的输出结果进行 shuffle 操作（见图 3）之后传递给 Reduce 函数。如图 4 所示，Reduce 函数根据相同 key 对应的 value 中的首个字符分别对 a 和 b 且序列号相同的元素相乘，并将同一个 key 下得到的一系列乘积结果进行叠加，最终得到原始大矩阵相乘的所有元素计算结果。

通过 Hadoop MapReduce 平台下矩阵乘的并行化设计，可以分别并行化计算出求解$\hat{\beta}$的式(3)中两个核心的矩阵乘：$\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 以及 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T}$。

3.2　基于 Spark 的矩阵乘法并行化

基于 Spark 的矩阵乘并行化设计与基于 Hadoop 的矩阵乘并行化设计的基本思想比较类似，主要差别为基于 Hadoop 的实现主要考虑 Map，Reduce 函数及〈key，value〉键值对的设计，基于 Spark 的实现则主要考虑 RDD 的构建、分片及并行处理。以 3.1 节中的矩阵相乘为例，基于 Spark 的矩阵乘并行方案如图 5—图 7 所示。由图 5—图 7 可知，在基于 Spark 的矩阵乘并行化中，将矩阵 A 的行以及矩阵 B 的列作为 value，而将矩阵乘所得元素的坐标作为 key 生成 PairRDD，然后通过 Spark RDD 的操作 API——转换（transformation）ReduceByKey 得到矩阵 $A\times B$ 的各个元素，即将 A 的行向量与 B 的列向量进行点乘最终得到矩阵乘的各个元素值。

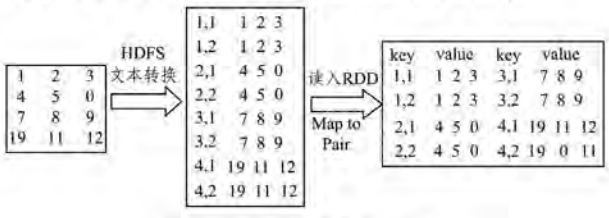

图 5　矩阵 A 转换为 PairRDD

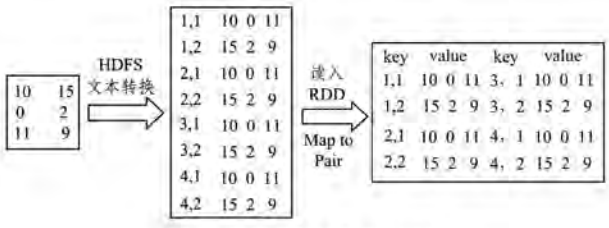

图 6　矩阵 B 转换为 PairRDD

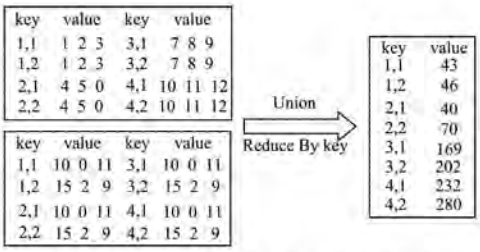

图 7　通过 RDD 转换得到矩阵乘积的元素

通过上述基于 Spark 的矩阵乘并行化设计，可以分别并行化计算出求解$\hat{\beta}$的式(3)中两个核心的矩阵乘：$\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 以及 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T}$。

3.3　高维矩阵求逆的并行化

在 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 为高维度矩阵的情况下，很难通过单机完成 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 矩阵的求逆工作，因此需要通过矩阵求逆的并行化方案进一步保障 ELM 算法在海量数据下的有效运行。LU 分解法可以并行化求解高维矩阵的逆矩阵[12]。

对于一个低维矩阵 $\boldsymbol{A}$ 而言，通过 LU 分解有：$\boldsymbol{A}=\boldsymbol{PLU}$，其中 $\boldsymbol{L}$ 为 LU 分解所得的下三角矩阵，$\boldsymbol{U}$ 为 LU 分解所得的上三角矩阵，$\boldsymbol{P}$ 为 LU 分解所得的置换矩阵。由此可以得到 $\boldsymbol{A}^{-1}=\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}^{-1}=\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$。

对于高维矩阵而言，利用 LU 分解并行化求逆时需要先将高维矩阵进行分块，将块标号作为标记并将块存储于分布式文件系统中；然后对每个主对角线上的块进行 LU 分解并将分解结果广播至集群的各个节点，在各个节点更新其他相关块，从而得到高维矩阵的 LU 分解结果；然后利用 $\boldsymbol{A}^{-1}=\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$ 求解矩阵 $\boldsymbol{A}$ 的逆矩阵，其中上三角矩阵 $\boldsymbol{U}$ 以及下三角矩阵 $\boldsymbol{L}$ 依然是高维度矩阵，需要通过将上（下）三角矩阵拆分成两个上（下）三角矩阵以及一个“矩形矩阵”；然后通过迭代求解两个上（下）三角矩阵的逆矩阵并利用求解结果更新“矩形矩阵”，将更新后的“矩形矩阵”以及两个上（下）三角矩阵的逆矩阵合并得到 $\boldsymbol{U}^{-1}$ 以及 $\boldsymbol{L}^{-1}$；最后利用 $\boldsymbol{A}^{-1}=\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$ 得到高维矩阵 $\boldsymbol{A}$ 的逆矩阵[12]。

3.4　ELM 模型训练并行算法

基于矩阵乘的并行化设计，给出 ELM 模型训练并行算法的流程图，如图 8 所示。

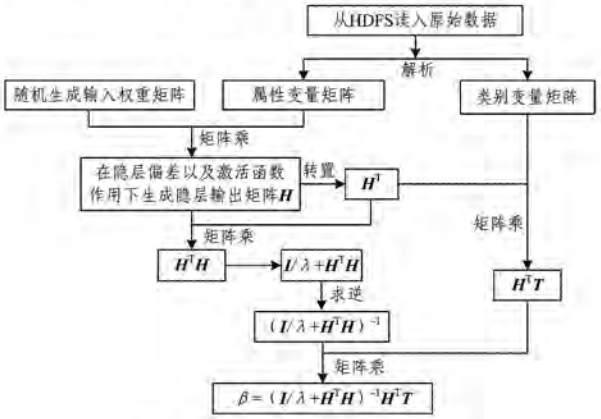

图 8　ELM 模型训练并行算法流程图

从图 8 可知，ELM 算法中的主体运算即为矩阵乘运算，

流程图中的矩阵乘均采用本文所设计的矩阵乘并行化方案。当 ELM 中的 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 为低维度矩阵时，图中的矩阵求逆运算可通过单机实现；若 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 为高维度矩阵，则利用 LU 分解法并行化求解其逆矩阵[12]。转置运算，非常简单，只需通过单机运算变换矩阵的行列坐标即可。至此，极限学习机 ELM 算法的并行化工作基本完毕。

4 实验设计与分析

本文实验平台集群包括 1 个主节点和 3 个从节点，硬件配置如表 1 所列。集群运行 Hadoop 为 Hadoop-1.2.1，Spark 版本为 Spark-1.5.2-bin-hadoop1.2.1，采用 Java1.7 版本编写所有 Hadoop 及 Spark 程序。本文实验所选用数据集为 iris 数据集、CarEvaluation 数据集、Adult 数据集以及 PowkerHand 数据集，具体如表 2 所列。

表 1 实验集群的硬件配置

Node	CPU/Core	Memory/GB	Network/(GB/s)
Master/Slave1	12	32	1
Slave2	2	8	1
Slave3	2	6	1

表 2 实验所用数据集

数据集名称	样本个数	特征个数	类别变量个数
Iris	150	4	1
CarEvaluation	1728	6	1
Adult	30000	14	1
PowkerHand	300000	10	1

本文共设计了 3 组实验，分别研究隐层节点数目、激活函数、训练样本占总样本的比例以及程序运行平台对 ELM 并行算法性能的影响。由于 ELM 算法随机选取隐层参数，为了排除因随机选取参数带来的测试误差，每次实验进行 5 次，去掉最大值和最小值，最终求 3 次测试结果的平均值。需要说明的是，本文实验结果只能保证在所选数据集上有显著意义，在其他数据集上的实验结果可能会有不同程度的差异，欢迎读者做比较实验并与作者探讨。

4.1 隐层节点数目对 ELM 预测准确率的影响

本组实验将隐层节点数目作为影响因素，主要研究其对预测准确率的影响。选取 Iris 数据集，分别将隐层节点数目设为 3,5,6,7,9，并分别选择常用的 sigmoidal 函数、RBF 函数以及正弦函数作为激活函数，实验每次所测得的准确率为 3 次运行的平均准确率。根据实验所测数据绘制出隐层节点数目-预测准确率曲线图，如图 9 所示。

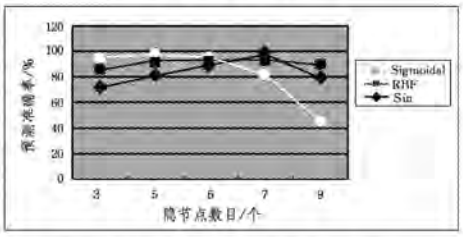

图 9 隐层节点数对预测准确率的影响

由图 9 可知，ELM 预测准确率在一定的隐层节点数目范围内比较稳定，当隐层节点数在 3～7 个时预测准确率相对稳定，基本处于 80%～100%之间，大多数情况都处于 90%以上，从而证明了只要隐层节点数目选取合适，ELM 算法具有较高预测准确率。纵向比较 3 个激活函数，sigmodal 函数的预测准确率受隐层节点变化的影响是三者中最大的，但是当隐层节点数目在 3～6 个时，sigmoidal 函数的预测准确率相比其他两个激活函数更高，基本达到 95%以上，而随着隐层节点数目的增长，其预测准确率下降明显，因此对于 sigmoidal 激活函数而言，选取合适的隐层节点数目非常重要；对于 sin 函数而言，其预测准确率受隐层节点数目变化的影响相对于另外两个激活函数最小；对于 RBF 激活函数，在隐层节点数目由 3 个逐渐增加至 7 个时其预测准确率呈上升趋势，但当隐层节点数目继续增加时，预测准确率开始降低，在隐层节点数目为 5～9 个时，其预测准确率达到 80%以上，因此对于 RBF 而言，其隐层节点数目为 5～9 个比较适合。

另外，从图 9 可以看出，对于 Iris 数据集而言，当隐节点数多于 7 个时，3 个激活函数所对应的预测准确率均呈现下降趋势，其原因是 Iris 数据集(样本数为 150)的数据量较小，当隐节点数目大于 7 个时会使得模型过于复杂从而导致过拟合，因此隐节点数不宜过多。针对 ELM 算法过拟合问题的有效解决办法有待继续研究。

本组实验说明在进行 ELM 并行算法模型训练时，隐层节点数目并不是选取得越多越好，而是需针对不同的样本数据量、不同的激活函数选取各自适合的隐层节点数目区间，避免因隐层节点数目的选择不当而使预测准确率降低。

4.2 训练样本比例对预测正确率的影响

本组实验将训练样本占总样本的比例作为影响因素，研究其对预测准确率的影响。实验所选数据集为 CarEvaluation 数据集，分别取样本训练比例为 1/6,1/2 和 5/6，隐层节点数目选为 6，分别测试在 Hadoop、Spark 以及单机中运行所得的正确率。

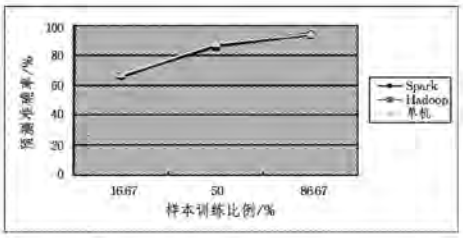

图 10 训练样本比例对预测准确率的影响

由图 10 可知，随着训练样本比例的提高，预测准确率基本呈线性增长，当训练样本比例达到 50%以上时，预测准确率达到 85%以上；当训练样本比例达到 83%以上时，预测准确率达到 95%以上甚至接近 100%，由此可见训练样本比例的增加对预测准确率的提升效果十分明显。另外，纵向比较基于 3 个不同平台的 ELM 算法的预测准确率变化曲线可知，由于基本算法原理相同，三者的预测准确率非常接近，即不同的程序运行平台对预测正确率的影响很小。

4.3　不同并行平台的 ELM 运行效率比较

本组实验所选数据集为 Iris 数据集、CarEvaluation 数据集、Adult 数据集以及 PowkerHand 数据集，如表 2 所列。隐层节点数目均为 6 个。对 4 个数据集分别做不同预处理并进行模型训练，分别于 Hadoop 以及 Spark 平台上运行所设计的程序，比较二者在运行效率上存在的差异。根据实验所测数据绘制条形统计图，如图 11 所示。

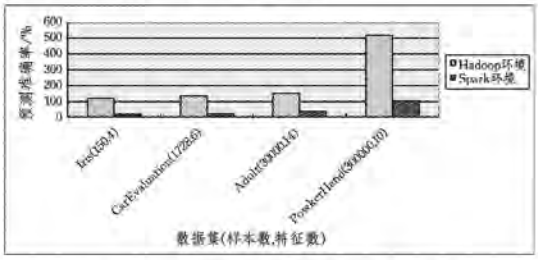

图 11　Hadoop 和 Spark 环境下的效率比较

如图 11 所示，随着样本数据量的增加，对数据并行化的处理时间的增长并不与数据量的增大呈线性关系，而是缓慢增长，如基于 Hadoop MapReduce 对 Iris 数据集（样本数为 150，特征数为 4）进行模型训练并验证所需时间为 112s，基于 Spark 运行时间为 17s；对 CarEvaluation 数据集（样本数为 1728，特征数为 6）进行模型训练并验证的时长分别为 117s，18s；对 Adult 数据集（样本数为 30000，特征数为 14）进行模型训练并验证所需时长分别为 151s，36s；对 PowkerHand 数据集（样本数为 300000，特征数为 10）进行模型训练并验证所需时长分别为 522s，109s。处理上述 4 个数据集的时间增长速度均远落后于数据量的增长速度。同时，可明显发现随着样本数据量的增加，基于 Hadoop 和 Spark 完成同样的训练以及验证工作所需时间的差值变大，即在进行大数据模型训练及验证时，样本数据量越大，相比于 Hadoop，Spark 的处理效率就越高。

结束语　本文分别设计并实现了基于 Hadoop 和 Spark 两大主流大数据处理平台的并行极限学习机算法，着重在矩阵乘法的并行化计算方面做了创新工作。实验表明，相比于传统的 Hadoop 平台，新一代基于内存计算的 Spark 平台在大数据量处理方面具有明显优势。另外，本文也对极限学习机中隐层参数选取的“随机性”问题进行了初步研究，证明隐层节点数目的确定与样本集及激活函数的选取都有关系。下一步将以 Spark 并行平台为主，在极限学习机的并行化力度、隐层参数确定及 kernel-ELM 并行化等领域继续进行深入研究。

参 考 文 献

[1] HUANG G B, ZHU Q Y, SIEW C K. Extreme learning machine: theory and applications[J]. Neurocomputing, 2006, 70(1): 489-501.

[2] HUANG G B, WANG D H, YUAN L. Extreme learning machines: a survey [J]. Int. J. Mach. Learn. & Cyber, 2011, 2(2): 107-122.

[3] HUANG G B, DING X J, ZHOU H M. Optimization method based extreme learning machine for classification[J]. Neurocomputing, 2010, 74(1): 155-163.

[4] HE Q, SHANG T F, ZHUANG F Z, et al. Parallel extreme learning machine for regression based on MapReduce[J]. Neurocomputing, 2013, 102(1): 52-58.

[5] 安俊秀，王鹏，靳宇倡. Hadoop 大数据处理技术基础与实践[M]. 北京：人民邮电出版社，2015：15-45.

[6] 王晓华. MapReduce2.0 源码分析与编程实践[M]. 北京：人民邮电出版社，2014：21-60.

[7] CHEN J, CHEN H, WAN X Y, et al. MR-ELM: a MapReduce-based framework for large-scale ELM training in big data era [J]. Neural Computing & Applications, 2016, 27(1): 101-110.

[8] 夏俊鸾，刘旭辉，邵赛赛，等. Spark 大数据处理技术[M]. 北京：电子工业出版社，2015.

[9] Pentreath N. Spark 机器学习[M]. 蔡立宇，等译. 北京：人民邮电出版社，2015：32-56.

[10] LIU Z Q, GU R, YUAN C, et al. Review of the parallelization of the classification algorithm based on SparkR [J]. Journal of Frontiers of Computer Science and Technology, 2015, 9(11): 1281-1294. (in Chinese)
刘志强，顾荣，袁春，等. 基于 SparkR 的分类算法并行化研究[J]. 计算机科学与探索，2015，9(11)：1281-1294.

[11] FERRARI S, STENGEL R F. Smooth function approximation using neural networks [J]. IEEE Transactions on Neural Networks, 2005, 16(1): 24-38.

[12] HUANG Y H, GU R, GAO X K. The method of parallelization of the computing of the inverse matrix of distributed dense matrix based on Spark: China, CN 105373517 A[P]. 2016-03-02. (in Chinese)
黄宜华，顾荣，高兴坤. 基于 Spark 的分布式稠密矩阵求逆并行化运算方法：中国，CN 105373517 A[P]. 2016-03-02.