Sparse Linear Algebra in the Deeplearning4j Framework

Thèse n. 1234 2011
présenté le 12 Mars 2011
à la Faculté des Sciences de Base
laboratoire SuperScience
programme doctoral en SuperScience
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Paolino Paperino

acceptée sur proposition du jury:

Prof Name Surname, président du jury
Prof Name Surname, directeur de thèse
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur

Lausanne, EPFL, 2011

Wings are a constraint that makes
it possible to fly.
— Robert Bringhurst

To my parents. . .

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Lausanne, 12 Mars 2011 D. K.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Key words:

Contents

List of Figures

List of Tables

Introduction

A non-numbered chapter...

Nowadays Machine learning is very popular and widely-used to resolve daily life problem

- deeplearning, neural net, self driving car etc

To work correctly and expect a accurate result, those machine learning problems require a huge amount of data. Such datasets are challenging regarding the execution time of the algorithms, the memory space required, the network usage when working imn a distributed environnement, etc

A big part of those problems used sparse datasets. For example a recommender system typically works with a dataset high-sparsity. This dataset contains the rating of movies or products given by the users. But usually the user only rated a very small subset of products.

That means, that if we know that our dataset will be sparse, we can used the sparse linear algebra to resolve the problem to optimized the memory used and the executing time.

Deeplearing4j didn't support any sparse format for vectors, matrices or tensors, neither the operations.

1 Sparse Data and Formats

Definition

Data are said sparse when it is contains only a few non-null values. That kind of dataset are really common in Machine Learning application and can be an high influence on the computation.

The sparsity of a dataset is defined by :

$$sparsity = \frac{\text{\# non-null values}}{\text{\# values}} \tag{1.1}$$

Conversely when a dataset has only a few null values, the data are said dense. The density of the dataset is defined by the inverse of the sparsity:

$$density = \frac{1}{\text{sparsity}} \tag{1.2}$$

Using dense methods and data structure with sparse data could have a severe bad impact on the performance

The Advantages of Sparse Data

Sparsity is a very useful property in Machine Learning. Some algorithms can have fast optimization, fast evaluation of the model, statistical robustess or other computational advantages. A lot of machine learning application are using sparse dataset such as recommender system, natural language processing algorithm,

Sparse Data are very common in Machine Learning

In Machine Learning it's very common to deal with sparse dataset. We can encounter them in any kind of applications: Natural Language Processing, Retrieving Systems, Recommender Systems, etc.

Given the possible optimization that sparse dataset allows and the high number of people that could take advantages of it, it becomes important to add the support of sparse data in Nd4j.

A Real Case of Sparse Dataset

In 2008 Netflix launched a contest, the Netflix Grand Prize [net()], to improve their recommender system model and to increase the accuracy of predictions and published an sample dataset made with the ratings of anonymous Netflix customers. The dataset had more than 100 millions sampled ratings and it contained about $m = 480'186$ users and $m = 17'770$ movies [Koren(2009)]. If stored as a dense matrix, it would need to store $8'532'905'220$ values in memory. That corresponds to a sparsity $\cong \frac{100'000'000}{8'532'905'220} = 0.011719338$.

Storing more than 8 trillions 64-bit floating-point numbers needs more than 64 gigabyte of memory which quickly become unmanageable even for the world's fastest supercomputers.

Solution: Encode the data into a Sparse Format

To avoid the high volume of storage needed issue, we must store the data into a sparse format. There are different kind of formats which each of them is more suitable to different aspects (storage, matrix-matrix operations, vector-matrix operations,...)

Formats

There exists several different formats to store a sparse array. The idea behind using a sparse format instead of the classic dense one, is to reduce the memory space and the executing time of the operations. Knowing that a matrix is sparse allows to shortcut some operation steps. For example during a matrix multiplication, we can avoid to perform the multiplication for the zero elements of the sparse matrix.

Matrices

Coordinates Format

This format is the simplest format to encode a sparse array. The coordinates and the value of each non-zero entry are stored in arrays. Typically each element are encoded in a tuple (row, column, value)

Some implementation variations of the COO format exist. The elements can be sorted along a dimension, or it can be some duplicate indexes.

$$A_{(M\times N)} = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 3 \\ 1 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} \quad \rightarrow \quad \begin{aligned} Values_{(1\times NNZ)} &= \begin{bmatrix} 2 & 3 & 1 & 4 \end{bmatrix} \\ Rows_{(1\times NNZ)} &= \begin{bmatrix} 0 & 1 & 2 & 2 \end{bmatrix} \\ Columns_{(1\times NNZ)} &= \begin{bmatrix} 1 & 2 & 0 & 2 \end{bmatrix} \end{aligned}$$

Figure 1.1: A matrix stored in COO format

With this format it's easy and fast to retrieve the value given an index and to insert a new non-zero element.. It's also fast and simple to convert into a dense format.

But this format don't minimize the memory space. It can be reduced with a compressed format such as CSR or CSC as described below.

Compressed Row Format

The Compressed Row and the Compressed Column formats are the most general format to store a sparse array. They don't store any unnecessary element. But it requires more steps to access the elements than the COO format.

Each non-zero element of a row are stored contiguously in the memory. Each row are also contiguously stored.

The format requires four arrays:

Values	All the nonzero values are store contiguously in an array. The array size is NNZ.
Column pointers	This array keeps the column position for each values.
Beginning of row pointers	Each pointer i points to the first element of the row i in the values array. The array size is the number of rows of the array.
End of row pointers	Each pointer i points to the first element in the values array that does not belong to the row i . The array size is the number of rows of the array.

Compressed Column Format

The Compressed Column Format is similar to CSR but it compresses columns instead of rows.

Given a matrix $N \times M$, the pointers arrays will have a size M.

$$A_{(N\times M)} = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 4 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix} \rightarrow \begin{aligned} Values_{(1\times NNZ)} &= \begin{bmatrix} 2 & 3 & 1 & 4 & 2 & 1 \end{bmatrix} \\ Columns_{(1\times NNZ)} &= \begin{bmatrix} 1 & 2 & 0 & 2 & 2 & 3 \end{bmatrix} \\ pointersB_{(1\times N)} &= \begin{bmatrix} 0 & 1 & 2 & 2 & 4 \end{bmatrix} \\ PointersE_{(1\times N)} &= \begin{bmatrix} 1 & 2 & 2 & 4 & 6 \end{bmatrix} \end{aligned}$$

Figure 1.2: A matrix stored in CSR format

Tensors - Multi-dimensional arrays

A tensor is a multi-dimensional array. The order of the tensor is the dimensionality of the array needed to represent it. Matrices and vectors can be represented as tensors where the order is equals to 2 and 1 respectively.

This generalization allows a more generic implementation of a n-dimensional array in the Nd4j library.

Coordinates Format

The COO format can easily be extended to encode tensors by storing an array of indexes instead the row and column coordinates.

A array of order $K = 3$ with shape $N \times M \times P$ which has the following non-zero values :

value	indexes
1	0 1 0
2	1 1 2
3	1 2 0
4	2 0 1
5	2 2 0

can be encoded with one values array and one indexes array :

$$\begin{aligned} Values_{(1\times NNZ)} &= \begin{bmatrix} 1, & 2, & 3, & 4, & 5 \end{bmatrix} \\ Indexes_{(NNZ\times K)} &= \begin{bmatrix} [0,1,0], & [1,1,2], & [1,2,0], & [2,0,1], & [2,2,0] \end{bmatrix} \end{aligned}$$

Figure 1.3: A tensor stored in COO format

2 The Deeplearning4j Library

Deeplearning4j is a open-source Deep Learning library for the JVM. It runs on distributed CPU's and GPU's.

Architecture of the library

The library is composed by several sub-libraries:

Deeplearning4j provides the tools to implement neural networks and build computation graphs

Nd4j is the mathematical back-end of Deeplearning4j. It provides the data structures for the n-dimensional arrays and allow Java to access the native libraries via JavaCPP and the Java Native Interface.

Libnd4j is the computing library that provides native operations on CPU and GPU. It's written in C++ and Cuda.

Datavec provides the operations for the data processing such that data ingestion, normalization and transformation into feature vectors.

The Importance of Nd4j in the Library

Nd4j is at the base of the Deeplearning4j library, it provides data storage, manipulations, and operations. It gives the atomic pieces needed to build more complex deep learning systems such as neural networks. Nd4j stands for N-Dimensional Arrays for Java and is basically a scientific computing library for the JVM. It features n-dimensional array object and the support of CPU and GPU via Cuda.

The APIs provided by the library are essentially wrappers for the different version of BLAS (Basic Linear Algebra Subprogram).

BLAS is a specification that defines the low-level routines for linear algebra operations (for vectors and matrices). There exist several libraries implementing those subroutines in C or Fortran for dense or sparse formats. In Nd4j the BLAS subroutines can directly be called from Java thanks to JavaCPP, that internally uses the Java Native Interface (JNI) to call native routines from the JVM environment. This architecture allows the library to benefit from the advantages of the native side.

Nd4j needs a Sparse Representation

Currently in Deeplearning4j Sparse Data are treated like dense and use the dense operations of BLAS and Libnd4j to perform computations. With a new sparse representation we could gain in storage space and computation speed.

3 Structure of an Multi-dimensional Array

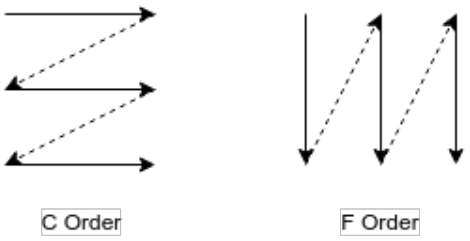
The new sparse array have to be compliant with the API and inter-operable with the current dense array implementation.

Let's start by studying how the dense array is made of.

Storing an Array

A dense array is stored as a single contiguous block of memory, flatten in a one-dimensional array. Arrays are stored off-heap (outside the JVM environment). The reasons behind this design decision are numerous : better performance, better interoperability with BLAS libraries, and to avoid the disadvantages of the JVM such as the limited size of arrays due to the integer indexing (limited to $2^{31}-1 \cong 2.14$ billion elements)

There are two methods to store a multi-dimensional array into a linear memory space: row-major order (C) or column-major order (Fortran). Figure 3.1 shows how a two-dimensional array is stored according to the order.



$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} C-Order &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} \end{bmatrix} \\ F-Order &= \begin{bmatrix} a_{11} & a_{21} & a_{12} & a_{22} & a_{31} & a_{13} \end{bmatrix} \end{aligned}$$

Figure 3.1: Comparison between C-order and F-order

The data are accessed via strides which define how to index over contiguous block of data. For

each dimension it defines by how many values two consecutive dimensions are separated. In the case of the matrix A defined in figure 3.1, the strides would be $(3,1)$ in case of C-order and $(1,3)$ in case of F-order. Strides $(3,1)$ means that each row is separated by 3 values and each column is separated by 1 value.

Data Buffer

Databuffer is a storage abstraction. This allows for backend optimal storage

Information concerning the shape of the array

..

Hierarchy of Arrays

..

Important Component of the Arrays

..

Indexes

..

Views

..

Operations

..

4 Implementation

Limitations due to the Current Library

DataBuffers have a fixed length

CSR Matrices

Structure

Get or Put Data into this format

Limits with this format

COO Tensors

First implementation

More parameters are needed to define the tensors

All and Interval Indexes

Point Index

Specified Index

New Axis Index

Computations of the the Parameters

Computation of the Sparse Offsets

Computation of the Flags

Computation of the Hidden Dimensions

Sparse Indexes Translation

..

5 Operations

Backends

BLAS

Level 1 in CSR Matrix

Level 1 in COO Tensor

Libnd4j

..

6 Results

..

l

..

7 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

A An appendix

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Bibliography

[net()] Netflix grand prize. http://www.netflixprize.com.

[Koren(2009)] Yehuda Koren. 1 the bellkor solution to the netflix grand prize, 2009.