Handwriting Recognition Using a Cloud Runtime

Kathleen Ericson
Colorado State University
1100 Center Avenue
Fort Collins, CO 80523
970-492-4126
ericson@cs.colostate.edu

Shrideep Pallickara
Colorado State University
1100 Center Avenue
Fort Collins, CO 80523
970-492-4209
shrideep@cs.colostate.edu

Charles W. Anderson
Colorado State University
1100 Center Avenue
Fort Collins, CO 80523
970-491-7491
Chuck.Anderson@colostate.edu

ABSTRACT

Handwriting recognition can be a difficult task – while there are many commercial products in place which can be trained for individual users, these programs often have problems abstracting patterns learned in order to work for multiple users. In this work we use the Optical Recognition of Handwritten Digits [3] dataset from the UCI Machine Learning Repository, which contains handwritten digits from 43 people. Using basic neural network code written in R, we analyze different approaches to classification in a distributed environment: Through R itself with Snowfall [7], and with Granules [8, 9] using JRI [1] as well as a binary bridge for communication to the same base R code.

Keywords

Handwriting recognition, neural networks, cloud, Granules, MapReduce.

1. INTRODUCTION

Artificial Neural Networks (ANNs), inspired by the connection of neurons in the brain, can handle the modeling of complex data. As an interpreted language designed for fast matrix multiplications, R [2] is an ideal language for neural networks. Fully training a large neural network to handle a complex dataset can, however, take hours to complete. One approach to achieve greater accuracy more quickly is to partially train several smaller networks and use their combined output to develop an expert classification. A distributed approach can be used to fully take advantage of this speedup – the training occurs in parallel, so all neural networks should complete training at approximately the same time.

There are several R packages designed for parallel computing [10, 11], though many have a steep learning curve, and require modification of serial code. Our goal in this paper is to compare distributed implementations which can all use the same serial code backend. To do so, we decided to use the R package Snowfall [7]. It allows the user to send serial code to several different machines in a cloud, and does not require an extensive background in parallel computing to fully leverage. We will also be using the light-weight cloud computing runtime Granules [1] for comparison. Granules allows a user to specify how and when a computation is run in the cloud, and can allow a computation to enter a dormant state between runs, allowing the computation to retain state between runs. Additionally, Granules has been designed to handle streaming data, so the computation can go dormant while waiting for more data to become available – meaning that network setup can be considered a one-time cost. Granules is Java-based, so we need to use a tool in order to bridge between Java and R. We have used two different approaches, in order to analyze the different overheads introduced. We are using the Java R Interface (JRI) [1] as well as a byte-based communication protocol over sockets.

In this paper we are specifically looking at the application of neural networks to classify handwritten digits from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/). This dataset contains handwritten digits that have been scanned in, and then interpreted as a 16x16 bitmap. This is a real-world example, which could benefit from the ability to handle streaming data – digits are scanned in as needed, sent to the cloud to be classified and the computation can return to a dormant state once finished.

In our approach, each machine in the distributed environment is responsible for training a small neural network, and then using it to classify any data sent to it. Once all the small networks have finished classification, the results are sent to a single node which is then responsible for generating an expert classification based off of these results.

For all of our experiments we are interested in gathering data on overheads introduced by distributing computations at each step. We are interested in how long it takes with each approach to set up a distributed network and use it, as well as any overheads introduced for communication in this environment. To get the closest to baseline readings, we are using small neural networks, and not taking accuracy into account. Additionally, we initially train a single neural network in each environment in order to isolate communications overheads as the networks grow. While we expect a small communications overhead to appear as more nodes are added to the cloud, all training and classification should be handled in parallel and stay close to our single-node tests. In these tests, we are comparing three different environments:

1. Baseline times in R – this consists of running the original sequential code on a single node.
2. Snowfall times – the code is distributed among a cluster, where each node is responsible for a single neural network.
3. Granules times – calls are made from the Java runtime down to the original R code on each machine, each test is run using both JRI to bridge between Java and R, as well as our binary bridge across sockets.

The rest of this paper is laid out as follows – Section 2 details related work in this field. Section 3 describes the layout of the

Snowfall and Granules networks, as well as information about the dataset we are using, and how this data is used for classification. In section 4, our experiments are laid out, and the results are presented. Section 5 discusses our conclusions, and plans for future work.

2. Related Work

R has many applications in both statistical analysis and machine learning, and is designed for the fast implementation of code. This makes it an ideal language for machine learning approaches such as artificial neural networks. While there are several R packages for distributed computing, these packages often involve modification of serial code, have limited options for computation lifetime, and can have a steep learning curve. In this work we use Snowfall for distributed computing.

Snowfall is an extension of Snow [11], one of the first distributed packages for R. It has a simple user interface, and better load balancing capabilities than Snow. Both packages have been designed to run sequential code in parallel in a distributed environment. We are essentially using Snowfall to follow the Map-Reduce paradigm.

As mentioned above, Granules is a lightweight cloud computing runtime designed for spreading thousands of computations across available machines. Granules allows a user to specify computations which follow the MapReduce framework or dataflow graphs [6]. It has been designed to handle streaming data from sensors and has been deployed in several applications ranging from gathering data from GPS sensors to detect tectonic movements to BCI applications [5]. These incoming data streams and any associated processing necessary can be interleaved on a single machine – a single machine may be handling thousands of computations simultaneously.

Granules computations can be scheduled to run a specified number of times, at a given interval, or as data available. A user can specify run semantics along any of these axes, and can create as complex a run policy as the situation demands. Furthermore, these policies can be changed at runtime in order to adapt to changes in policy or environment. Across every run, a computation builds and maintains state – a useful trait when dealing with computations with a high start-up cost, such as training a neural network.

Hadoop [12] is another MapReduce framework built on Java. Hadoop has run-once semantics, and is file-based. Having the same Java backend as Granules, we should see the same communication overheads for our bridging techniques in Hadoop as we gathered in our Granules tests. Granules allows us to train a neural network, and then maintains the network between successive testing runs – entering a dormant state when there is no data to analyze. Hadoop, however, does not support streaming datasets, and the run-once semantics mean that we would need to write the trained neural network to file, and load it back up again when we needed to classify data – this would add an extra overhead for all classification runs that would be dictated by disk speeds.

3. Data and Network Design

In our experiments, we are looking at Granules clouds as well as Snowfall clusters. Both approaches involve training neural networks on individual resources/nodes in the network, having each node forward on their individual predictions to an intermediate step which aggregates predictions, and then returns a final classification to the user. In the Granules version, we have used the MapReduce paradigm [4] to separate out these steps: Each Mapper holds a neural network, and pushes classifications to the Reducer, which returns the expert classification based on the results of the mappers.

The Snowfall approach closely mimics this: The user pushes data along with trained neural networks out to the cloud, where each node is responsible for classifying the data with the neural network passed to it. This is then returned to the user, where a local equivalent of the reduce function is run in order to find the expert opinion. At each step, the same R code is executed in the backend.

The remainder of this section describes the data we have used for our tests, as well as the setup of the individual neural networks – including how the data shapes the networks.

3.1 Data

In these experiments we used the Optical Recognition of Handwritten Digits dataset from the UCI Machine Learning Repository. This data consisted of a training and test set, of 3823 and 1797 samples respectively. Each digit (0-9) is sampled approximately the same amount of times. The handwritten digit is scanned in, and divided into a 16x16 grid. The level of shading in each grid is interpreted as a double, this is then stored in a 1x256 array. Samples were gathered from 43 people: 30 for the training set, and 13 for the test set.

3.2 Neural Network Design

For these experiments, we used a simple neural network with 4 hidden units, which has gone through 400 iterations of training. While this is a relatively small number of hidden units and abrupt training schedule, with multiple smaller networks we can fully leverage a distributed approach, and return an "expert" opinion after gathering results from all the individual smaller networks.

Each network has 256 inputs (corresponding to the digit bitmap inputs), 4 hidden units, and 10 outputs – each neural network determines the probability of the input being each of the 10 digits, and will return a prediction of the number with the highest probability.

4. Experiments

This section contains the results of our experiments run in a sequential R environment, a distributed but still pure R version using Snowfall, and the hybrid Java/R Granules implementations. For the hybrid approach, we tested with both JRI and Granules byte-based communication approaches. The R code base used in all versions is exactly the same, with only the calling approach varying slightly across all 3 versions, due to interface differences.

4.1 Training baselines

The first point we need to address is the amount of time needed to initialize and train a neural network. For these tests, we loaded the training dataset, then randomly sorted the elements – this prevents the neural networks from learning patterns such as a long stretch of 1s will occur immediately after a long stretch of 0s.

While we expect that this will add some extra variance in our tests, we still expect training times to remain relatively consistent.

4.1.1 R training baselines

For these tests, we simply looked at the basic amount of time needed to initialize and train a single neural network using our original sequential code. The results for this can be seen in Table 1.

Table 1 Training times (ms) for 1NN in pure R

Mean	Min	Max	SD
95711.97	86562.47	100427.4	2831.515

From these results, we can see that it takes, on average, 95 seconds to fully train a neural network with our base, sequential code. When looking at the Granules results, however, it appears that the method with inherent overheads is achieving better results. We believe that this is a result of a difficult arrangement of the training set, and plan to look into this discrepancy in the future.

4.1.2 Snowfall training baselines

The Snowfall approach again gathers data on the time it takes to initialize and train a neural network. For these tests we ran with a single neural network on a single machine as well as 10 networks spread across 10 machines. The results of these runs are shown below in Table 2.

Table 2 Training Times (ms) with Snowfall

#NN	Mean	Min	Max	SD
1	182983.4	171508.1	194573.2	4381.567
10	177956	150463	193839.1	12767.57

At first, the faster training time across 10 neural networks is confusing, but having a different ordering of training sets is most likely the cause of this. What we can definitely see is that we are achieving approximately the same training times across these approaches – meaning that we are parallelizing training effectively, and there is no great overhead introduced when adding more nodes to the Snowfall cluster. We can also clearly see that a greater variance is introduced as more nodes enter the configuration. As more nodes are added, we start seeing network delays in our communication overhead.

4.1.3 Granules Training baselines

For the Granules tests, we again ran on both single machines and 10 machines. We ran all tests with both JRI and our byte-based bridge for Java/R communications. The results are below in Table 4 and Table 3 respectively.

Table 3 Training Times (ms) with Granules (JRI)

#NN	Mean	Min	Max	SD
1	94059.06	85422	101669	3397.396
10	129007.1	101779	187987	12333.08

Table 4 Training Times (ms) with Granules (BRIDGE)

#NN	Mean	Min	Max	SD
1	76003.76	69900	82890	2497.206
10	95847.32	78297	138165	6877.097

Both Granules approaches are achieving a mean training time below that of the pure R baseline. We believe this to be a result of a fortuitously arranged training input, but still showing a minimal overhead introduced by bridging communications between Java and R. Another point to note is the increase in standard deviation as we move from a single instance to 10 neural networks spread across 10 different machines: the byte-based bridge shows that variance increased by 3, while the JRI approach increased by 4 times. As both are using Granules to handle communications, it seems to show effects of scaling on the underlying bridges.

4.2 Streaming Classification Times

For these experiments, we again tested with a base R version, a Snowfall R version, and the Granules JRI and Bridge versions. With the exception of the basic R tests, every test was run across 10 machines, each with an individually trained neural network. While the baseline R times will be from a single machine, we feel that this gives us a hard limit on our potential speed in a networked environment. For all tests, a single image was streamed for classification.

Table 5 Baseline R Classification Times (ms)

Mean	Min	Max	SD
0.2844	0.2611	0.4249	0.0262

Table 6 Distributed Classification Times (ms)

Method	Mean	Min	Max	SD
Snowfall	13132.3	13048	13419.83	83.43438
Granules (JRI)	122.4538	85.434	149.8	10.45
Granules (Bridge)	169.7102	125.2334	213.7027	28.05974

While the Snowfall version essentially had a hard limit of 13 seconds, the Granules approaches both fell well under the second mark. They are both still, however, far slower than the pure R single machine version. In an attempt to track down these discrepancies, we also tried running the Granules versions with a single neural network, but did not find significantly different results. We are hitting a hard limit with our communications protocols.

5. Conclusions and Future Work

From our work, we can clearly see that Granules is an effective method of distributing R computations across a cloud. With our binary bridge, we are outperforming JRI with respect to training our distributed neural networks, but falling behind in classifying speed. While the Granules approaches introduce a heavy overhead for classifications, we believe that the benefits gained from a distributed approach can outweigh these overheads – at less than 200ms response times, this approach can theoretically be used in a real-time system. We can clearly see that the Granules approaches can outperform the native R Snowfall distributed implementation.

The current round of tests focused on measuring overheads, and determining any penalties incurred by our bridging methods. Now that we have determined the best method for distributing our computations, a next step in our work is to focus on determining the best setup to maximize accuracy – we can modify both the number of nodes in the cloud, as well as the number of hidden units in each neural network.

In future work, we also plan on focusing on improving accuracy by passing more data between nodes. In the current implementation, each neural network is simply returning the single prediction it found most likely. We may be able to achieve better accuracy by instead returning the list of probabilities for each digit. This way the reducer would have more data to work with – and may be able to make a more informed final prediction.

6. REFERENCES

[1] JRI - Java/R Interface, 2009.

[2] The R Project for Statistical Computing, 2010, R project homepage.

[3] Alpaydin, E. and Kaynak, C. Optical Recognition of Handwritten Digits. University, D.o.C.E.a.B. ed., UCI Machine Learning Repository, Istanbul, 1998.

[4] Dean, J. and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *ACM Commun.*, *51*. 107-113.

[5] Ericson, K., Pallickara, S. and Anderson, Charles W. Analyzing Electroencephalograms Using Cloud Computing Techniques *CloudCom 2010*, Indianopolis, USA, (To appear) 2010.

[6] Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D., Dryad: distributed data-parallel programs from sequential building blocks. in *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, (Lisbon, Poutugal, 2007).

[7] Knaus, J. snowfall: Easier cluster computing (based on snow), 2010.

[8] Pallickara, S., Ekanayake, J. and Fox, G., Granules: A Lightweight, Streaming Runtime for Cloud Computing With Support for Map-Reduce. in *IEEE International Conference on Cluster Computing*, (New Orleans, LA., 2009).

[9] Pallickara, S., Ekanayake, J. and Fox, G., An Overview of the Granules Runtime for Cloud Computing. in *IEEE International Conference on e-Science*, (Indianapolis, 2008).

[10] Rosenberg, D.S. HadoopStreaming: Utilities for using R scripts in Hadoop streaming, 2010.

[11] Tierney, L., Rossini, A.J., Li, N. and Sevcikova, H. SNOW: Simple Network of Workstations, 2009.

[12] White, T. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.