Integration of Big Data Processing Tools and Neural Networks for Image Classification

Nikita E. Kosykh
Emperor Alexander I St. Petersburg State Transport University,
Saint Petersburg, Russia
nikitosagi@mail.ru

Anatoly D. Khomonenko
Emperor Alexander I St. Petersburg State Transport University,
Saint Petersburg, Russia
khomon@mail.ru

Alexander P. Bochkov
Peter the Great St. Petersburg Polytechnic University,
Saint Petersburg, Russia
kostpea@mail.ru

Anatoly V. Kikot
Emperor Alexander I St. Petersburg State Transport University,
Saint Petersburg, Russia
a.v.kikot@yandex.ru

Abstract

The issues of joint use of tools for processing big data in solving problems of artificial intelligence are becoming increasingly important. The article discusses the task of optimizing the parameters of neural networks used for image recognition using Matlab and Hadoop systems, as well as the MNIST (Modified National Institute of Standards and Technology) database is a voluminous database of handwritten number samples. The results of calculating the optimal number of neural network layers for solving the classification problem of images presented. We study the issues of evaluating the accuracy of image classification depending on the number of network neurons, choosing the optimal network training algorithm, and evaluating the effect of parallelization[Sha09] using MATLAB Distributed Computing Server in the process of training a neural network on computing performance.

1 Introduction

[1]The issues of joint use of tools for processing big data[Bah15] in solving problems of artificial intelligence are becoming increasingly important

In: A. Khomonenko, B. Sokolov, K. Ivanova (eds.): Selected Papers of the Models and Methods of Information Systems Research Workshop, St. Petersburg, Russia, 4-5 Dec. 2019, published at http://ceur-ws.org.

Deep learning of neural networks is a very popular topic these days, especially in computer vision applications. Until 2010, there was no database[Liu16] sufficiently complete and voluminous in order to train a neural network in a high-quality way to solve certain problems related to image recognition. Existing solutions had a high level of inaccuracy. With the emergence of open databases, such as Kaggle (big data bank) and ImageNet (image bank), it is possible to train neural networks and make practically error-free decisions. The article discusses the task of optimizing the parameters of neural networks used for image recognition using Matlab and Hadoop systems.

Justification of the choice of programming environment. The Matlab 2016a[Kra18] chosen as an integrated programming environment for working with big data. It supports some tools for solving problems, namely:

- Variables stored on the hard disk. Using the matfile function, you can access MATLAB variables directly from the MAT file on disk, without loading the entire variable into memory. This will allow processing of large data sets that do not fit in memory.
- Datastore. The ***datastore*** function allows you to access data that does not fit in memory. This may include data from files, sets or tables.
- Parallel computing. Parallel Computing Toolbox implements parallel loops to run MATLAB code on multi-core architectures.

Machine learning. Machine learning used to develop predictive models for big data. The whole range of machine learning algorithms is contained in the Statistic Toolbox and Neural Network[Pao89] Toolbox modules.

- hadoop. Using the MapReduce[Gha15] and DataStore functionality built into MATLAB, you can develop algorithms on a personal computer and run them on Hadoop. You can request a piece of data using the ***datastore*** function, and then using the Distributed Computer Server, run the algorithms within the Hadoop MapReduce environment on the complete set of data.

2 MNIST Dataset of Digit Patterns

The MNIST (Modified National Institute of Standards and Technology) database is a voluminous database of handwritten number samples. The database is a standard proposed by the US National Institute of Standards and Technology for the purpose of calibrating and comparing image recognition methods using machine learning, primarily based on neural networks.

Neural networks tend to learn better from specific examples. In the development of neural[Nov15] network for pattern recognition, we use the popular MNIST handwritten data set (Hadoop, 2016). Kaggle's open portal for distributing big data [Peh19] uses this very kit in the Digit Recognizer training contest. The set contains the following components:

a) trainSet.csv – training data;
b) testSet.csv – test data for presentation.

First you need to load the training data into MATLAB (MATLAB, 2018). For this we use the built-in function ***csvread***.

```
M = csvread (filename, R1, C1)
trainSet = csvread (trainSet.csv, 1,0)
testSet = csvread (testSet.csv, 1,0)
trainSet = csvread ('trainSet.csv', 1,0)
subSet = csvread ('testSet.csv', 1, 0);
```

The first column in the ***trainset*** set is a label that shows the correct number for each sample in the data set, and each line is a sample. In the remaining columns, the row is an image of the handwritten digit 28x28, but all the pixels are placed in one row, and not in the original rectangular shape. To render the numbers, we need to rebuild the rows into 28x28 matrices. To do this, you can use the ***Reshape*** function, with the exception that you need to transpose the matrix, because the ***Reshape*** function works in columns and not line by line.

To display an image, you need to create a graphic context (window) using the function figure. As parameters, you can pass properties for a graphic context to a function. Initially handwritten numbers are displayed in colors that are presented in a color palette of green and blue shades (Fig. 1).

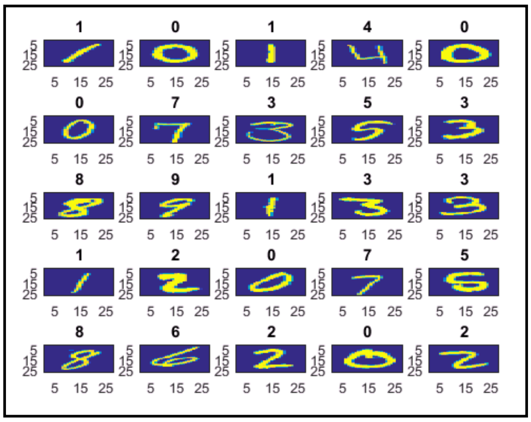

Figure. 1: Display the default set

For a better perception, we use the ***colormap*** () function, which sets the color map of the final image.

Colormap (gray) – sets a linear palette in shades of gray.

The following code fragment reads the first 16 lines of the data set, which are 16 digits. Converts rows into matrices and displays the resulting images on the screen, while the numbers on the top show the object class numbers.

```
figure
colormap(gray)
for i = 1:25
subplot(5,5,i)
   matrix = reshape(trainSet(i, 2:end),
   [28,28])'
imagesc(matrix)
title(num2str(tr(i, 1)))
end
```

After the above code fragment executed, an image of 16 handwritten numbers will appear on the screen, placed on a single graphic screen in a black and white palette (Fig. 2).

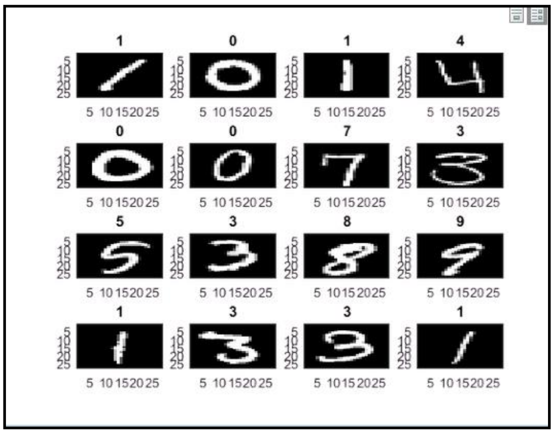

Figure. 2: Image handwritten numbers

To build the network[Kho15], we will use the tool for pattern recognition 'nprtool' from the Neural Network Toolbox module.

3 Data preparation

The input tool expects two sets of data:

a) ***input*** – a numeric matrix, each column of which represents samples and rows. These are scanned images of handwritten numbers;

b) ***vectorLabels*** – matrix-row binary form of 0 and 1, which is mapped to specific labels that represent the image. It is also called a dummy variable. Neural Network ToolBox also expects tags to be stored in columns and not in rows.

Class labels range from 0 to 9, as there are exactly so many single-digit numbers. To solve this problem, we will use "10" instead of "0", because MATLAB indexing starts from 1.

The ***trainSet*** dataset stores sample images in rows, not columns, so the entire dataset needs to transpose. The test set of data does not come out in advance in the MNIST set. To form it, we will hold 1/3 of each data set (training, test).

```
n = size(trainSet, 1);
labels   = trainSet(:,1);
labels(labels == 0) = 10;
labelsd = dummyvar(labels);
inputs = trainSet(:,2:end)
inputs = inputs'; % transposition of the
set of predictors
labels = labels';
labelsd = labelsd';
rng (1);
c = cvpartition (n, 'Holdout', n / 3);
XtrainSet = inputs (:, training (c));
YtrainSet = labelsd (:, training (c));
XtestSet = inputs (:, test (c));
YtestSet = labels (test (c));
YtestSetd = labelsd (:, test (c));
```

Explanation: The function *c = cvpartition (n, 'Holdout, p)* randomly creates a section for validating validation during observations. This section divides the array into tutorials and a test set. Parameter p must be a scalar. When 0 <p <1, *cvpartition* randomly selects n cases for the test set. The default is p = 1/10.

4 Using the Neural Network Toolboox

The following describes the steps to work with the Matlab NNTool.

A. To start working with NNT, call the ***nprtool*** method from the command window.

B. In the next menu, select Pattern Recognition Tool to open the pattern recognition tool.

C. On the welcome screen, go to "Select data".

D. For the input, choose XtrainSet and for classes, YtrainSet.

E. After installing the arrays, go to the "Data for verification" section. In this case, you can leave the default values. This will divide the data in the ratio of 70-15-15 into sets for training, testing and testing.

F. In the network architecture section, change the value of hidden layers to 100 and go on.

G. For training, go to the Train Network section and click "Train" to start training. Upon completion of the operation, a window with learning results will open. Numerical indicators can viewed as graphs and charts.

H. On the last tab, you will prompted to save the learning script for the model you just created. (eg NeuralNetworkSctipt.m).

Below is a diagram of a model of an artificial neural network (Figure 3.), which was created using the pattern recognition tool. It has 784 input neurons, 100 hidden layer neurons and 10 output word neurons (which is equal to the number of classes for prediction).

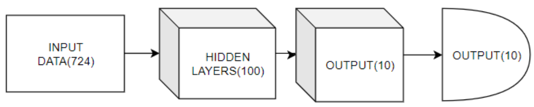

Figure. 3: Neural network model with parameters

The model trained by adjusting the weights for correct results. W in the diagram denotes weights and b – displacement neuron, which are part of individual neurons. Separate neurons in the hidden layer are as follows: 784 input neurons and correspondingly the same weights, 1 offset unit and 10 activation outputs.

5 Data Visualization

For begin you should look inside the structure of the ***NeuralNetworkSctipt.m*** file. There you can see the generated variables, such as IW1_1 and x1_step1_keep, which are weights vectors obtained in the process of training the network. Since we have 784 inputs and 100 neurons, the complete hidden layer will consist of a 100x784 matrix. Now you can clearly see what they study neurons. Copy the above variables from the file and enter them into the workspace in the form of matrices.

```
W1 = zeros (100, 28 * 28);
W1 (:, x1_step1_keep) = IW1_1;
Figure
colormap (gray)
for i = 1:16
subplot (4,4, i)
```

```
digit = reshape (W1 (i, :), [28,28])';
imagesc (digit)
end
```

6 Evaluation Of Classification Accuracy

Now you can use the previously prepared file to predict the classes in the part of the Xtest file held and compare them with the actual classes in the ***Ytest*** set '. This gives a real picture of the performance against the background of unclassified data. After that we will execute the commands that are from the 10x14000 matrix, convert to 1x14000 by selecting only the values with the maximum probability.

```
% predicts the probability for label
YpredSet =
neuralNetworkFunction(XtestSet);
% display the first 5 columns
YpredSet (:, 1: 5)
[~, YpredSet] = max(YpredSet);
```

It remains to compare the predicted network indices and actual. To do this, we apply the formula for assessing the quality of the algorithm, namely, assessing the accuracy of data classification.

$$Accuracy = \frac{P}{N},$$

where: **N** is the size of the training sample; **P** is the number of objects from the sample for which the network made the right decision.

This formula must have interpreted into MatLab code and our data sets, namely:

```
Accuracy = sum (YtestSet == YpredSet) / length (YtestSet) % compare predicted and actual

Accuracy = 0.9554.
```

The prediction accuracy rating is 95.5%, which is good enough for a network with a minimum number of manual settings.

7 Calculation The Optimal Number Of Layers

It now remains to find out how the change in the number of hidden neutrons will affect the accuracy of data classification. The main thing is to fulfill the inequality

$$Inputs \geq hiddens \leq outputs$$

In the previous experiment, 100 neurons of the hidden layer were involved, we will try to find the optimal value at which the accuracy factor will improved, but there will be no effect of reconfiguring the system.

As input parameters, we will set the number of neurons in the hidden layer. The amount will vary from 10 to 300 in increments of 25. All values will written in the layers array. Further, in the loop, it sets the values and the results of the accuracy of the network estimate and writes them into the Accuracy variable. Let as save a set of commands to the ***nnscript.mlx*** script, so that we can repeat requests.

```
layers = [15,50:30:290];
scores = zeros(length(layers), 1);
models = cell(length(layers), 1);
for i = 1:length(layers)
    hiddenLayerSize = layers(i);
    nn = patternnet(hiddenLayerSize)
    nn.divideParam.trainRatio = 0.7;
    nn.divideParam.valRatio = 0.15;
    nn.divideParam.testRatio = 0.15;
    nn = train(nn,XtrainSet,YtrainSet);
    p = nn(XtestSet)
    models{i} = nn,
    [~,p] = max(p)
    AccuracyScores(i) =
sum(YtestSet==p)/length(YtestSet)
end

figure
plot(layers, AccuracyScores, 'o-')
xlabel('Number of neurons')
ylabel('Accuracy of classification')
title('Number of neurons / accuracy')
```

As a result, of the execution of a sequence of commands, we obtain a graph of the dependence of the accuracy of the algorithm on the number of neurons in the hidden layer (fig. 4).

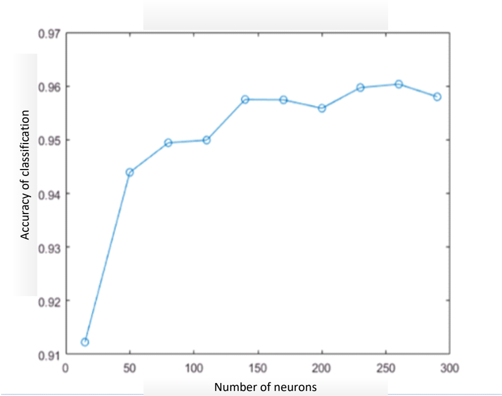

Figure. 4: The dependence of the classifier accuracy on the number of neurons

Analyzing the graph, we can conclude that the best result will be about 145 neurons, with an accuracy of 0.957, then, with an increase in the number of neurons in the hidden layer, the accuracy remains the same, and performance begins to decline markedly.

Note that we obtain greater accuracy with an increase in the number of neurons, but at some point the accuracy may fluctuate in the negative direction (due to the accidental initialization of the weights). As the number of neurons increases, the model can capture more functions, but because of their excess, you can eventually retrain your model on one set, and this will have a bad effect on the classification of new data.

Consider the question of justifying the choice of the optimal learning algorithm.

8 Selection of the optimal learning algorithm

The following algorithms were chosen for network training, which are presented in Table 1. The preliminary selection was made on the basis of the performance studies of algorithms for solving various typical problems by the MathWorks group.

Table 1: Neural Network Learning Algorithms

Function Matlab	Algorithm
'trainscg'	Stochastic Gradient Descent
'trainrp'	Resilient Propagation (Rprop)
'traincgf'	Fletcher-Powell related gradient method
'traincgb'	Powell-Bill related gradient method
'traincgp'	Polac-Ryber method of associated gradients
'trainoss'	One-step algorithm of the cutting planes method

```
funcArray={'trainscg','trainrp','traincg
b','traincgp','traincgf','trainoss'}
```

All computational processes were performed on a single personal computer without the use of distributed computing, i.e. without additional optimization.

```
model = cell(length(funcArray),1)
accuracyRate = cell(length(funcArray),
2);
for i = 1:length(funcArray)
net = patternnet(145,funcArray{i,1})
net.divideParam.trainRatio=0.7
net.divideParam.valRatio=0.15
net.divideParam.testRatio = 0.15;
rng(1);
tic % timer start
net = train(net,XtrainSet,YtrainSet)
toc% timer stop
timer1 = toc
model{i} = net
p = net(XtestSet);
[~,p] = max(p)
accuracyRate{i,1} = funcArray{i}
accuracyRate{i,2}=sum(YtestSet==p)/leng
th(YtestSet)
accuracyRate{i,3} = timer1
end
```

First, an array of cells is created to store future models of trained networks. Each model contains a trained network on a specific algorithm. Next, we select the variable that will store the accuracy estimates of the algorithm, the name of the algorithm, and the network training time

The code in the loop performs network training using a constant number of layers. Data for training is divided in the classical proportion 70/15/15. We loop through the data into the array of accuracyRate cells.

To get the results in a time-sorted form, execute the following command:

sortrows (accuracyRate, [3])

The test results of learning algorithms with no parallel computing [39] are presented in table. 2

Table 2: Algorithm Test Results for Conventional Computing

Learning Algorithm	Classification Accuracy	Learning Time, sec.
'trainrp'	0.9038	71.0844
'trainscg'	0.9566	149.0134
'traincgp'	0.9558	249.0219
'traincgb'	0.9589	313.0135
'traincgf'	0.9612	409.3543
'trainoss'	0.9588	845.4352

The first column contains the names of the algorithms, the second – the accuracy of data classification, the third network training time in seconds. As we can see, the ***Rrop*** turned out to be the fastest method for learning, but the accuracy is poor. The optimal approach, in terms of execution time and accuracy, is stochastic gradient descent with a result of 95.6%.

9 Distributed Learning Computing

Parallel Computing Toolbox allows training and building a neural network using multiple processor cores on a single PC or on multiple network computers using the MATLAB Distributed Computing Server.

Using multiple cores can speed up calculations. Using multiple computers can solve the problem of lack of RAM to accommodate too large data sets for one computer.

The goal is to use the tool and identify patterns between the number of cores involved and the network learning rate on the above algorithms.

To manage cluster configurations, the Cluster Profile Manager is used.

To open the pool of MATLAB workers, enable the default cluster profile, which refers to the local CPU core, use the following command:

```
pool = parpool
```

You must also indicate the number of workstations involved, or in our case the cores, by calling the command:

```
pool.NumWorkers
```

Now we can train the neural network by sharing data among the CPU cores. To do this, set the parameters for the training and testing network functions.

```
net = train(net, XtrainSet,
YtrainSet, 'UseParallel,' Yes ')
p = net(XtestSet, 'UseParallel,'
Yes ');
```

By starting the module using the 'ShowResources' argument, you can verify that the calculations are performed on several cores.

```
net = train(net, XtrainSet,
YtrainSet, 'useParallel', 'yes',
'showResources', 'yes')
p = net(XtestSet, 'useParallel',
'yes', 'showResources', 'yes');
```

MATLAB indicates which resources were used.

When the training and testing methods of the network are called, they divide the input data into distributed composite values, after performing the operations, they transform the data back into an array view into the original representation in the form of a matrix or an array of cells.

Here are the results of comparing the performance of computing the same algorithms for training a neural network, but using two physical cores to parallelize operations (Table 3).

Table 3: Using Parallel Computing for Network Learning

Algorithm	Accuracy	Time, sec.
'trainrp'	0.9038	87.6
'trainscg'	0.9566	187.3
'traincgp'	0.9558	268.9
'traincgb'	0.9589	330.9
'traincgf'	0.9612	371.6
'trainoss'	0.9588	638.5

At least we will build a Matlab bar chart comparing the network training time using one core and two processor cores (Fig. 5) of a personal computer.

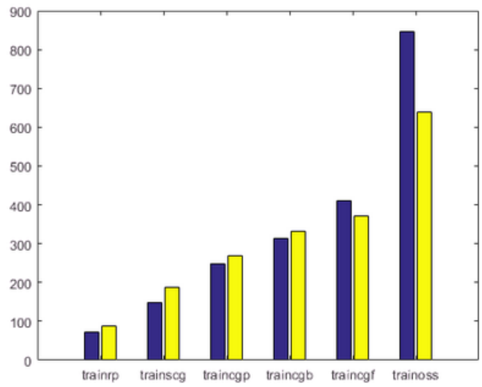

Figure. 5: Comparison of 2CPU and CPU performance

Note that the effect of using distributed computing becomes more noticeable when using algorithms with greater convergence and requiring more iterations to obtain a result.

A multilayer neural network has been built to solve the problem of machine learning, namely, optical recognition of handwritten characters based on the training data set MNIST.

In a number of experiments, optimal parameters (the number of hidden neurons, learning algorithm) were determined to achieve good accuracy of the data classification algorithm and network training time.

The ***trainrp*** function is the fastest pattern recognition algorithm. Its performance also deteriorates as the error value decreases. The memory requirements for this algorithm are relatively small compared to others.

In particular, ***trainscg***, the stochastic gradient descent algorithm, seems to have done a good job with a large number of weights. ***SCG*** works almost as fast with pattern recognition as ***RProp***, but performance does not decrease when error is reduced.

Other algorithms become very slowly with an increase in the number of neurons in the network, however, they can be useful in situations where a slower convergence of the function is required.

When using tools for parallel[Sha09] distributed computing, there is an increase in productivity in the learning speed for complex algorithms, based on connected gradients. Observations must be carried out with a minimum software load on the disk array; otherwise the results will vary greatly during idle and peak loads.

The stochastic gradient descent algorithm is optimal for the task of pattern recognition based on neural networks with an average number of input neurons. Unlike other algorithms, its performance does not decrease with a decrease in error.

When using tools for parallel distributed computing, there is an increase in productivity in the learning speed for modified algorithms, based on conjugate gradients.

At the preparatory stage, prior to building a neural network, the Apache Hadoop framework is deployed for the task of storing data in a pseudo-cluster and providing access to data through the Java interface.

A common advantage of the approach is its versatility and the ability to integrate with the existing infrastructure of the enterprise and its cloud storage, and services. Due to the abundance of existing libraries for working with neural networks and big data, the idea can interpreted for any high-level programming language with the appropriate qualifications of a programmer.

10 Conclusion

Matlab and Hadoop tool sharing technologies discussed above can find application for optimizing the process of using neural networks to solve various applied problems of artificial intelligence.

Acknowledgments

The work was partially supported by the grant of the MES RK: project No. AP05133699 "Research and development of innovative information and telecommunication technologies using modern cyber technical means for the city's intelligent transport system".

References

[Bah15] Bahrami M., Singhal M. The role of cloud computing architecture in big data. Information granularity, big data, and computational intelligence. Springer, Cham. Pp.275–295.

[Peh19] Pehcevski J. (2019). Big data analytics: methods and applications. Arcler Press. Canada. 430 p.

[Gha15] Ghazi M. R., Gangodkar D. Hadoop, MapReduce and HDFS: a developers perspective. Procedia Computer Science. Pp. 45–50.

[Kra18] Krasnovidov, A.V., Khomonenko, A.D., Zabrodin, A.V., Smirnov, A.V. On the peculiarities of the exchange of data between applications in high-level languages and Matlab functions. CEUR Workshop Proceedings. Workshop Computer Science and Engineering in the framework of the 5 th International Scientific-Methodical Conference "Problems of Mathematical and Natural-Scientific Training in Engineering Education. St. Petersburg, Russia, November 8-9, 2018. Vol. 2341, pp. 33-41.

[Liu16] Liu J. Rethinking big data: A review on the data quality and usage issues. ISPRS Journal of Photogrammetry and Remote Sensing. 2016. Vol. 115. Pp. 134–142.

[Nov15] Novikov P.A., Khomonenko A.D., Yakovlev E.L. Justification of the choice of neural networks learning algorithms for indoor mobile positioning. Proceeding CEE-SECR '15 Proceedings of the 11th Central & Eastern European Software Engineering Conference in Russia. Moscow, Russian Federation. October 22-24, 2015. ACM New York, NY, USA ©2015. Article No. 9.

[Sha09] Sharma G., Martin J. MATLAB®: A language for parallel computing. International Journal of Parallel Programming. 2009. Vol. 37. No 1. Pp. 3–36.

[Pao89] Pao Y. Adaptive pattern recognition and neural networks. Reading, MA: Addison-Wesley, 1989. 309 p.

[Sha09] Sharma G., Martin J. MATLAB®: a language for parallel computing. International Journal of Parallel Programming. 2009. Vol. 37. No. 1. – Pp. 3-36.

[Kho15] Khomonenko A. D., Yakovlev E. L. Neural network approximation of characteristics of multi-channel non-Markovian queuing systems. SPIIRAS Proceedings. 2015. Issue 4(41). Pp.81-93.