Computation Graphs use in the Neural Networks for Handwritten Digit Recognition

Ivaylo Penev and Milena Karova
Technical University of Varna
1 Studentska strVarna 9010, Bulgaria
{ivailo.penev@tu-sofia.bg} {mkarova@ieee.bg}

ABSTRACT: *Neural networks have promises for developing handwritten digit processing and hence in this paper, we have given model for training. In this work we have used computation graphs for training and measuring the neural network. The features of the propose system is that it speed up the network building and training processing, and resulted in the new network model in the runtime environment. We have explained how we build the computation graphs while developing neural network and training. We have carried out experimental tests and described the use of data for hand-written digits training.*

Keywords: Machine Learning, Neural Networks, Computation Graphs, Classification, Recognition, Parallel Calculation

Received: 13 March 2021, Revised 25 June 2021, Accepted 9 July 2021

DOI: 10.6025/jcl/2021/12/3/69-76

1. Introduction

Handwritten digit recognition is a famous problem with multiple applications in theory and practice. It is a classification problem, where the input data (images of handwritten digits) are classified to one of a set of classes (digits from 0 to 9).

Neural networks (NN) are classical method for recognition and classification. Many solutions of NN for handwritten digit recognition have been proposed. Many of them achieve accuracy of more than 99.7% (i.e. the error is lower than 0.3%) (e.g. [3], [4], [5]), but the following main disadvantages could be pointed out in them:

- The NN model is complex and therefore hard for scaling;
- The NN model is dependent of the implementation tools (libraries, frameworks);
- The NN training requires much time.

The NN building and training typically consists of lots of independent calculation operations. The natural approach for NN building and training is using parallel calculations via the available computing devices in many systems - multicore central processing units (CPUs) and graphic processors (GPUs). In some literature sources solutions, using GPU are proposed (e.g. [1], [2]). Other sources present the application of the MapReduce method for NN training (e.g. [7]). Although the published results are very good, the implementation is still specific to platforms and libraries. For example the implementation of a NN for a system CPU and GPU needs serious modification for a system with multicore CPU but without GPU support.

Recently have been developed tools (programming frameworks and runtime environments), which allow the building and training process of NN to be presented by computation graphs. In graph nodes are placed calculation operations, in the arcs are the data for the operations. The main goals are two - possibility for automatic distribution of the graph between different devices of the computing system and achieving of relative independence of the NN model from the runtime environment (architecture, programming languages and frameworks). Some of the popular frameworks, providing computation graphs, are Theano, Torch. In the last two years has grown the usage of TensorFlow, developed by Google [6].

The present paper describes NN for recognition of handwritten digits from 0 to 9. Building, training and estimating the network are implemented as computation graphs, suitable for parallel execution by different devices. Results from experimental tests of the trained network in a machine learning framework, providing computation graphs and parallel execution of the graphs, are discussed.

2. Neural Network Building and Training by Computation Graphs

A simplified model of a computation graph is shown on Figure 1. The operations are placed in the nodes and their operands (sets of calculation values) are placed in the arcs. The framework and runtime environment divide the graph into subgraphs and submit each subgraph to a device of the system.

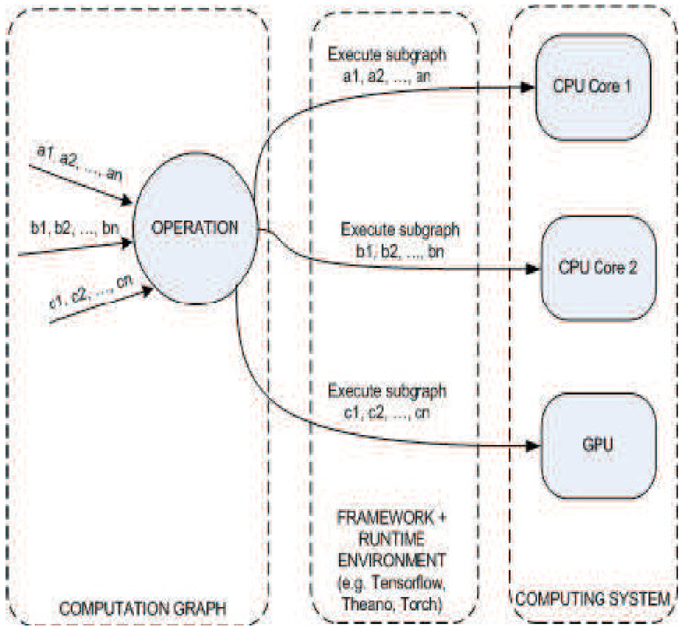
a1, a2, ..., an
b1, b2, ..., bn
c1, c2, ..., cn
OPERATION
COMPUTATION GRAPH
Execute subgraph a1, a2, ..., an
Execute subgraph b1, b2, ..., bn
Execute subgraph c1, c2, ..., cn
FRAMEWORK + RUNTIME ENVIRONMENT (e.g. Tensorflow, Theano, Torch)
CPU Core 1
CPU Core 2
GPU
COMPUTING SYSTEM

Figure 1. Simple computation graph, divided into subgraphs

2.1. Presentation of the Input Data

For training and testing the NN data sets from the MNIST data base with patterns of handwritten digits are used [8]. A digit image is presented as a two-dimensional matrix. The algorithm processes the digit as a set of 0s and 1s (Figure 2)

The two-dimensional array is converted to a vector, thus constructing the digit working set, suitable for algorithmic processing:

digit = {x1 x2 … xn}, where xi = {0,1}, 1 - available pixel in the digit image, 0 - not available pixel in the digit image.

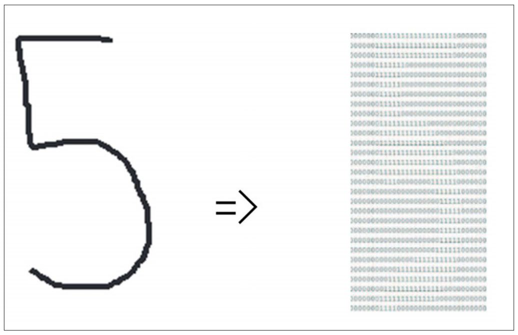

Figure 2. Presentation of an example digit from MNIST

2.2. Structure of the Neural Network

For building the NN model logarithmic softmax regression is used. The process follows the next two steps:

2.2.1. Calculation of Digit Distance to a Definite Class

The distance is a numeric value, estimating the similarity of a digit with a definite class (from 0 to 9). This value is calculated by a weighted sum of the intensities of the input digit image pixels. If a pixel has high intensity, which does not belong to a definite class, then the pixel weight is negative. Otherwise the pixel weight is positive.

For a class *i* the distance value is calculated as follows:

$$H_{y'} = -\sum_{i} y_i' \log(y_i) \qquad (1)$$

Where W_i - digit weights for a class *i*

b_i - biases of the digit for the class *i*

j – index for all pixels weights of digit *x*.

2.2.2. Calculation of Probability for belonging of the Digit to a Definite Class

The distance value (Eq. 1) is used for calculation of the probability, showing how likely is the jth digit from the training data set to belong to a class i from 0 to 9. The softmax function is used (Eq. 2):

$$H_{y'} = -\sum_{i} y_i' \log(y_i) \qquad (2)$$

The model of the neural network is constructed (Figure 3).

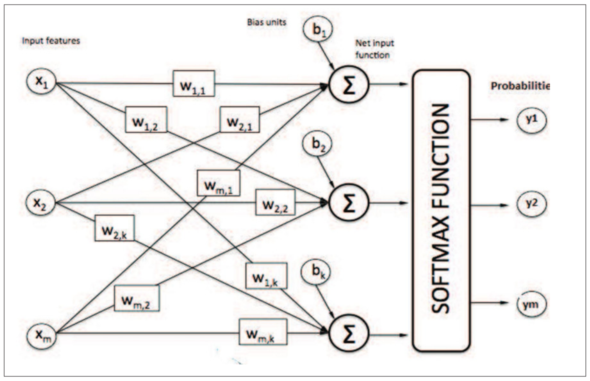

Figure 3. NN structure for handwritten digit recognition

The multiplication of the sets *W* and *x* consists of a lot of independent calculations for each data item from the training data set. These calculations could be performed in parallel. The computation graph of the *NN* model includes the possibilities for parallel calculation of *W* and *x* multiplication (Figure 4).

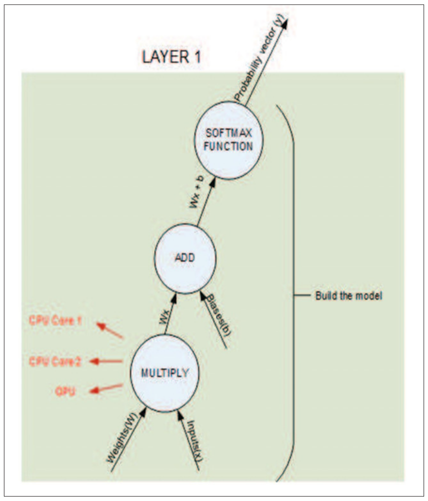

Figure 4. Computation graph of the NN building

2.3. Training of the Neural Network

For estimating the classification error cross-entropy function is used. For each digit from the training data set the error $H_{y'}$ is calculated (Equation 3)

$$H_{y'} = -\sum_i y_i^{'} \log(y_i) \qquad (3)$$

where

y_i - The calculated probability for digit to belong to class *i*,

y_i - The true class label of the digit, known from the training data set.

The *NN* training is performed by backpropagation and stochastic gradient descent algorithms. The purpose is finding the minimum of the $H_{y'}$ error by iteratively changing the values of the weighted parameters *W*.

The cross-entropy function consists of a set of independent calculations, which could be performed in parallel. At this step of the *NN* training a new vertex to the computation graph is added (Figure 5).

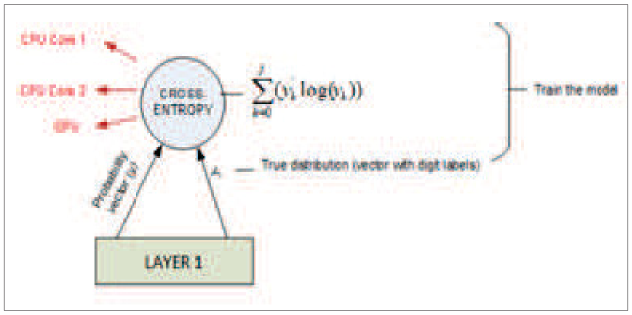

Figure 5. Computation graph for the NN training

2.4. Estimation of the Neural Network Accuracy

The accuracy is estimated using the training data sets. Each predicted value y is compared to the corresponding class label y' from the training data set. The result from the comparison is a set of values 1 (TRUE) if *y* is equal to *y*' and 0 (FALSE) otherwise. The average value of the equalities is finally determined as $\frac{\sum_j (y==y')}{j}$ for each jth item of the training data set.

For example the result [1 0 1 1] -> [TRUE FALSE TRUE TRUE] means accuracy of 0.75, i.e. 75%.

A new vertex for the accuracy estimation is added to the computation graph (Figure 6).

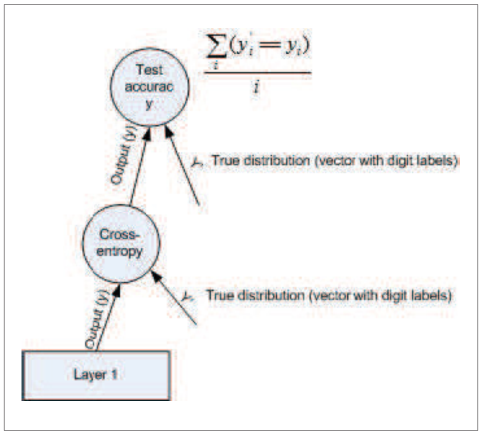
$\frac{\sum_i (y_i' = y_i)}{i}$
Test accuracy
Output (y)
True distribution (vector with digit labels)
Cross-entropy
Output (y)
True distribution (vector with digit labels)
Layer 1

Figure 6. Computation graph for NN accuracy estimation

3. Experimental Results

3.1. Framework for the Tests

The process of building, training and estimating the NN, described in the previous sections, is implemented by the TensorFlow framework, which provides Python-based programming tools and runtime environment for machine learning problems. The calculation operations and the data values for them (called tensors) are presented as computation graphs. The runtime environment automatically distributes the graphs to the devices of the system (CPU cores, GPU).

The experiments are performed on two computing systems - computer 1 (with GPU support) and computer 2 (without GPU support).

The TechPowerUp GPU-Z application is used for measuring the video card work in real time. The systemparameters are monitored by the Windows task manager.

3.2. Measuring the NN Training Time

For measuring the NN training two parameters are used: number of iterations – how many iterations are performed in the gradient descent algorithm for minimizing the error function;

batch size - number of data sets from MNIST, processed at each iteration of the gradient descent.

The data size for each test is determined as *data size = number of iterations * batch size.* For example a test with 500 iterations and 50 MNIST input digits is performed by data size of 500 * 50 = 25000.

The results show, that the NN training on a system with CPU and GPU reduces significantly the training time - about 8-10 times in comparison to a system with CPU only (Figure 7). When increasing the values of the test parameter values the difference between the times decreases, but the distributed computation graph is still calculated 6-8 times faster than the same graph on a CPU only (Figure 8).

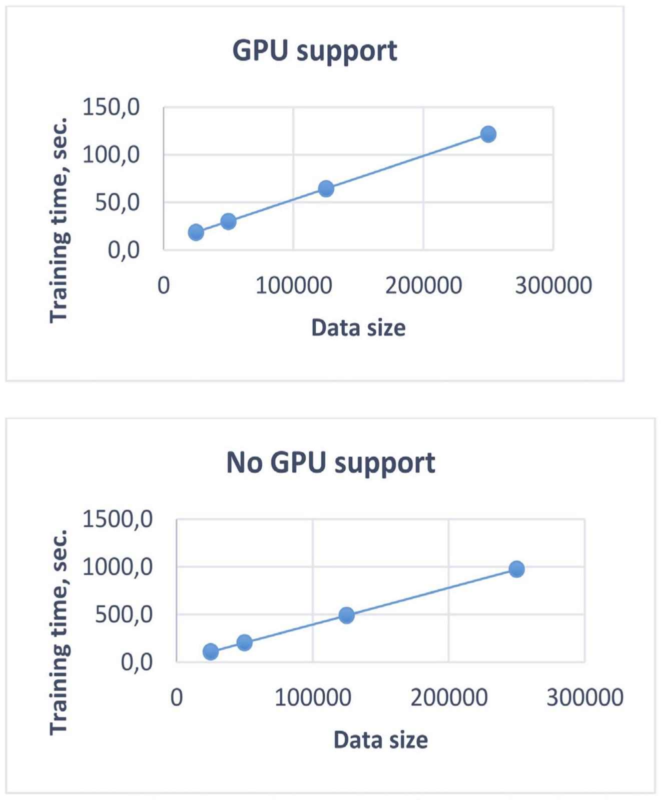

Figure 7. Training time with 500 iterations and 50 batch size

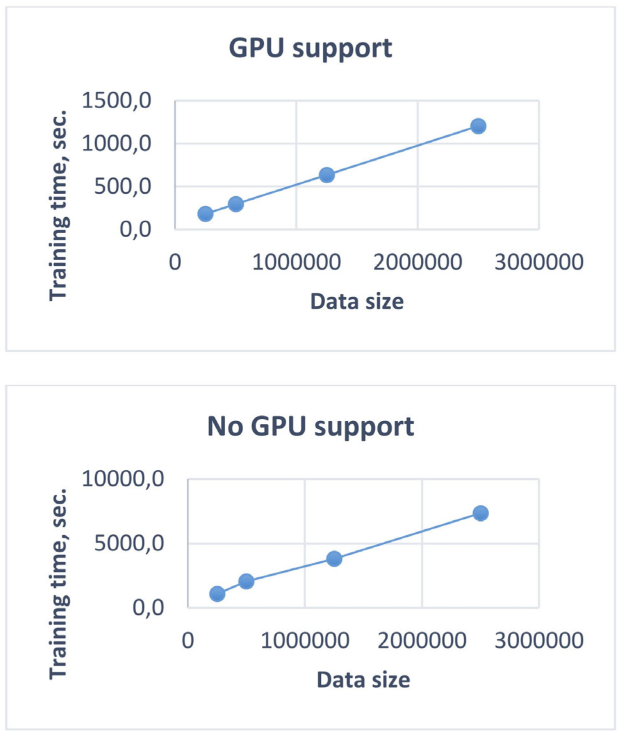

Figure 8. Training time with 5000 iterations and 250 batch size

3.3. Measuring the NN Training Accuracy

The best training accuracy achieved is 99.5% (Figure 9). This result is close to the best known results for handwritten digit recognition by NN. For example in [1] an accuracy of 99.65% is reported (i.e. error 0.35%), but the NN network is more complex with multiple hidden layers and is performed on a system with many video cards - precondition, which is not available in massive computer systems

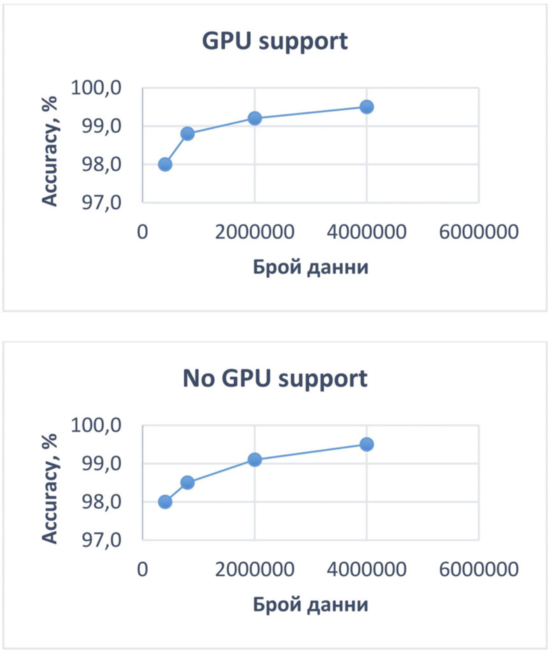

Figure 9. Training accuracy with 8000 iterations and 500 batch size

4. Conclusion

The following conclusions could be derived from the presented results:

- The building and training of neural network could be significantly accelerated by the usage of proper computation graphs, performed on systems with GPU;

- The presented approach could be used for other problems, because the process of NN building and training typically includes the same (or very similar) steps;

- NN, presented by computation graphs, could be adapted for implementation on other frameworks and environments, which provide distribution of the graphs to different system devices (e.g. frameworks as Theano or Torch).

The future work will be directed to application of computation graphs for various problems by other machine learning algorithms (e.g. kNN, SVM, etc.), where lots of independent calculation operations are typically performed.

References

[1] Ciresan, D., Meier, U., Gambardella, L., Schmidhuber, J. (2010). Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition, Neural Computation, Vol. 22, MIT, USA.

[2] Cire_an, D., Meier, U., Schmidhuber, J. (2012). Multi-column Deep Neural Networks for Image Classification, Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference, DOI: 10.1109/CVPR.2012.6248110, IEEE, 2012.

[3] Sato, I., Nishimura, H., Yokoi, K. (2015). APAC: Augmented Pattern Classification with Neural Networks, https://arxiv.org/abs/1505.03229, 2015.

[4] Chang, J.-R., Chen, Y.-S. (2015). Batch-normalized Maxout Network in Network, https://arxiv.org/abs/1511.02583, 2015.

[5] Chen-Yu, L., Gallagher, P., Zhuowen, T. (2015). Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree, https://arxiv.org/abs/1509.08985.

[6] Abadi, M., Agarwal, A., et. al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, arXiv:1603.04467v2.

[7] Basit, N., Zhang, Y., Wu, H., Liu, H., Bin, J., He, Y., Hendawi, A. (2016). MapReduce-based deep learning with handwritten digit recognition case study, Big Data (Big Data), 2016 IEEE International Conference, DOI: 10.1109/BigData.2016.7840783, IEEE.

[8] http://yann.lecun.com/exdb/mnist/