Demystifying Deep Learning Frameworks- A Comparative Analysis

1Divyanshu Sinha, 2JP Pandey, 3Bhavesh Chauhan
1 Department of Computer Science Engineering, GNIOT Gr Noida
2 Faculty of Electrical Engineering, KNIT, Sultanpur, India
3 Department of Electrical Engineering, ABESIT, Ghaziabad

1divv4u@gmail.com, 2tojppandey@rediffmail.com, 3blessu35@gmail.com
Corresponding Author: Divyanshu Sinha
Email: divv4u@gmail.com

https://doi.org/10.26782/jmcms.2019.02.00010

Abstract

Deep learning is a rapidly growing field of machine learning which finds the application of its methods to provide solutions to numerous problems related to computer vision, speech recognition, natural language processing, and others.

This paper gives a comparative analysis of the five deep learning tools on the grounds of training time and accuracy. Evaluation includes classifying digits from the MNIST data set making use of a fully connected neural network architecture (FCNN). Here we have selected five frameworks—Torch, Deaplearning4j, TensorFlow, Caffe & Theano (with Keras), to evaluate their performance and accuracy.

In order to enhance the comparison of the frameworks, the standard MNIST data set of handwritten digits was chosen for the classification task. When working with the data set, our goal was to identify the digits (0–9) using a fully connected neural network architecture. All computations were executed on a GPU.

The key metrics addressed were training speed, classification speed, and accuracy.

Keywords: Deep Learning, Feedforward MLP, Keras, Tensorflow, Theano, Caffe, Deaplearning4j, Torch

I. Introduction

As per [1] Muhammad Ramzan et al. fully connected neural network (FCNN) can be considered as a feedforward multilayer perceptron (MLP). Here
“feedforward” implies that data between the layers is sequentially transformed from input to output with no feedback connections: the output of a layer does not go to the input of the previous layers. (The presence of feedback connections introduces certain elements of dynamic behavior into a network; that is, the dependence of the output results on time.

In [II] Anuj Dutt et al. compares the results of some of the most widely used Machine Learning Algorithms like SVM, KNN & RFC and with Deep Learning algorithm like multilayer CNN using Keras with Theano and Tensorflow. Using these, he was able to get the accuracy of 98.70% using CNN (Keras+Theano) as compared to 97.91% using SVM, 96.67% using KNN, 96.89% using RFC.

In [III] Subhransu Maji et al. explores the use of certain image features, blockwize histograms of local orientations, used in many current object recognition algorithms, for the task of handwritten digit recognition. Existing approaches find that polynomial kernel SVMs trained on raw pixels achieve state of the art performance. Their approach achieves an error of 0.79% on the MNIST dataset and 3.4% error on the USPS dataset, while running at speeds comparable to the fastest algorithms on these datasets which are based on multilayer neural networks and are significantly faster and easier to train.

In [IV] Alexander K. Seewald, focuses on two major factors: the relationship between the amount of training data and error rate (with respect to the effort to collect training data to build a model with a given maximum error rate) and the transferability of models' expertise between different datasets. While the relationship between amount of training data and error rate is very stable and to some extent independent of the specific dataset used—only the classifier and feature representation have significant effect—it been impossible to transfer low error rates on one or two pooled datasets to similarly low error rates on another dataset. They have nicknamed this weakness as brittleness, inspired by an age old Artificial Intelligence term that means the same thing.

In our research, the term “fully connected neural network” is chosen to emphasize its difference from convolutional neural networks (CNNs): in adjacent layers of a FCNN, all connections between neurons are present, and information transmission is described by the full weight matrix $W(k)ij$ for the $k$ layer.
Figure 1 FCNN: An input layer (green), a hidden layer (blue), and an output layer (red)

The main aim of a FCNN is data classification. As per the widely known universal approximation theorem, any continuous function (which is, a classifier) can be denoted by a feedforward network with a single hidden layer containing a finite number of neurons.

The only issue is that the theorem does not impose an upper limit on the size of the single intermediate layer. Therefore, in practice we vary the number of intermediate layers instead.

Choosing a particular network architecture is defined by the type of data and requires tuning. Architectures having one intermediate layer and one neuron in it are equivalent to linear classifiers such as logistic regression.

Figure 2 The yellow area designates a FCNN subnet equivalent to a classifier such as logistic regression.
II. Training Data Set

As per [V] Li Deng MNIST data set comprises of handwritten digits for testing the five deep learning frameworks. The **MNIST database** (Mixed National Institute of Standards and Technology database) is a huge database of handwritten digits that is commonly used for training various image processing systems. This database is also used for training and testing in the area of machine learning. It was created by "re-mixing" the samples from NIST's original datasets.

![MNIST example](image)

**Figure 3** MNIST example

MNIST data set parameters:

- Image size: 28x28 pixels
- Number of images in the training set: 60,000
- Number of images in the test set: 10,000
- Number of class labels: 10
### Evaluation Methods and Metrics

We have compared the five deep learning frameworks when classifying the images of handwritten digits from the MNIST data set.

| Framework          | Technologies          |
|--------------------|-----------------------|
| Caffe              | C++, Python, Protobuf |
| Deeplearning4j     | Java, Spark, Hadoop   |
| TensorFlow         | Python, C++           |
| Theano             | Python                |
| Torch              | Lua                   |

**Table 1** Technologies used in the frameworks

The focus was mainly on measuring the convergence speed of neural networks of the FCNN type during the training stage and the classification accuracy of the trained networks during the prediction stage.

In particular, we computed the following metrics:

- Time of convergence
- Time of classification
- Accuracy of classification
- Amount of code necessary for writing the algorithm on a tested framework

The number of training epochs was set to one, five, and ten. The batch size was 128 images.

| Symbol | Metric |
|--------|--------|
| M1-m, M1-s | The average time of training with a fixed number of training epochs |
| M2-m, M2-s | The average time of testing with a fixed number of epochs |
| M3-m, M3-s | The accuracy of classification with a fixed number of training epochs |
| M4     | The amount of code needed for solving the problem |

**Table 2** The description of the evaluation metrics

150
Evaluation of metrics was performed on modifiable architectures of neural networks to achieve scalability of the frameworks during training and classification tasks. Two types of scaling were applied to the network architecture (see Figures 4.1 and 4.2):

- Varying the “depth” of the network (number of internal layers) with a fixed number of neurons in each layer
- Varying the “width” of the network (number of neurons) with a fixed number of layers

When modifying the network “depth,” the values of the control parameters used were as follows:

- Number of neurons in a single hidden layer: 100
- Number of layers were changing from 1 to 4

When modifying the network “width,” the following parameters were chosen:

- Network having one hidden layer was used.
- The number of neurons was being changed in the following way: 64, 128, 512, 1,024.

Figure 4 The evaluation of the main frameworks metrics depending on the network “depth”
Figure 5 The evaluation of the main frameworks metrics depending on the network "width"

For Testing of neural networks two different activation functions were used:

- Hyperbolic Tangent Function (Tan h)
- Rectified Linear Unit function (ReLU)

| Name  | Number of hidden layers | Neurons per hidden layer | Total adjusted network parameters |
|-------|-------------------------|--------------------------|----------------------------------|
| N1-L1 | 1                       | 100                      | 76,344                           |
| N1-L2 | 2                       | 100                      | 86,448                           |
| N1-L3 | 3                       | 100                      | 98,484                           |
| N1-L4 | 4                       | 100                      | 112,318                          |

Table 2 Architectural parameters on modification of network “depth”
### Table 3 Architectural parameters on modification of network “width”

| Name   | Number of hidden layers | Number of neurons in the hidden layer | Total adjusted network parameters |
|--------|-------------------------|---------------------------------------|----------------------------------|
| N2-N1  | 1                       | 64                                    | 47,345                           |
| N2-N2  | 1                       | 128                                   | 100,334                          |
| N2-N3  | 1                       | 512                                   | 403,646                          |
| N2-N4  | 1                       | 1,024                                 | 798,765                          |

### IV. Results & Analysis

**Effects of varying the network “depth”**

![Graphs showing effects of varying network depth](image)
Figure 6 Dependency of training time, testing time, and accuracy on change of network “depth” for the Tanh nonlinearity (1st, 5th, and 10th training epochs)
Figure 7: Dependency of training time, testing time, and accuracy on change of the network “depth” for the ReLU nonlinearity (1st, 5th, and 10th training epochs)
Effects of varying the network “width”

Figure 8 Dependency of training time, testing time, and accuracy on change of network “width” for the Tanh nonlinearity (1st, 5th, and 10th training epochs)
Figure 9 Dependency of training time, testing time, and accuracy on change of network “width” for the ReLU nonlinearity (1st, 5th, and 10th training epochs)
V. Complexity of Frameworks

The number of code lines necessary for implementing the algorithm used to train and test the FCNN architectures described in the above research is presented in Table 4.

| Framework            | Lines of code | Programming language |
|----------------------|---------------|----------------------|
| Caffe                | 110           | Protobuf             |
| Deeplearning4j       | 130           | Java                 |
| TensorFlow           | 80            | Python               |
| Theano (Keras)       | 62            | Python               |
| Torch                | 165           | Lua                  |

Table 4 Framework complexity in terms of lines of code

![Code complexity as the number of code lines](image)

Figure 10 Code complexity in terms of lines of code (the lesser, the better)
VI. Conclusion

From the results of the testing, the following conclusions can be drawn:

- It was observed that, as the network “depth” increases, it becomes necessary to increase the training time as well to sustain or improve the quality of recognition.
- When the size of a hidden layer (the number of neurons) is increased, it also requires an increase in the training time to keep or improve the quality of recognition.
- Use of nonlinear activation function ReLU leads to increase in the speed of training (compared to Tanh) and the resulting recognition accuracy.
- The deep learning frameworks under study have demonstrated quite similar results for the selected evaluation metrics on a GPU. The frameworks vary in classification accuracy and performance, hence there emerges no clearcut winner.
- Deeplearning4j was observed as the slowest framework during the network training and classification. It is important to take into consideration the fact that this framework is still under development and its training and classification speed may change in the near future.

At last after completion of testing the results on various parameters such as speed, classification accuracy, and the complexity of launching (the number of source code lines), the ranking list we got was: Theano (with Keras), TensorFlow, Caffe, Torch, and Deeplearning4j.

Having said the above, we need to remember that in general it is impossible to provide a fully objective comparison due to inevitable differences between the frameworks. For instance, while counting the necessary lines of source code, it is important to note that the interfaces of the frameworks under comparison use different programming languages, such as Python, Lua, or Java, and their lines of code are not equivalent.
References

I. Anuj Dutt, Aashi Dutt. "Handwritten Digit Recognition Using Deep Learning." International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 6, Issue 7, July 2017.

II. Alexander K. Seewald, “On the Brittleness of Handwritten Digit Recognition Models,” ISRN Machine Vision, vol. 2012, Article ID 834127, 2012.

III. Li Deng Microsoft Research, Redmond, Washington USA. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web] " IEEE Signal Processing Magazine ( Volume: 29 , Issue: 6 , Nov. 2012 )

IV. Muhammad Ramzan, Shahid Mehmood Awan, Hikmat Ullah Khan, Waseem Akhtar, Ammara Zamir, Mahwish Ilyas. "A Survey on using Neural Network based Algorithms for Hand Written Digit." International Journal of Advanced Computer Science and Applications, Vol. 9, No. 9, 2018

V. Subhransu Maji and Jitendra Malik EECS Department University of California, Berkeley Technical Report No. UCB/EECS-2009-159 November 25, 2009 http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-159.pdf