‘Learning convolutional Neural Network”

Beatriz Gómez Ayllón

April 5, 2017

Abstract

Add abstract

Contents

Contents **v**

I MOMORY **1**

1 Introduction **3**
1.1 Introduction . . . 3
1.2 Programming language and frameworks . . . 3
1.2.1 Python . . . 3
1.2.2 Theano and others frameworks . . . 3
1.3 Databases . . . 4
1.3.1 MNIST digit database . . . 5
1.3.2 Labeled faces in the wild . . . 5
1.3.3 FRAV dataset . . . 6
1.3.4 CASIA dataset . . . 7
1.3.5 MSU - MFSD database . . . 9

2 Methodology **11**
2.1 LeNet . . . 11
2.1.1 Structure of Lenet . . . 12
2.1.2 Results of LeNet . . . 12
2.1.3 Modifying LeNet . . . 17

Bibliography **21**

List of Figures **23**

List of Tables **25**

Part I

MOMORY

Chapter 1

Introduction

1.1 Introduction

This document blaaa blaaa

1.2 Programming language and frameworks

1.2.1 Python

Python is a object-oriented programming language and it is used to develop the experiments necessaries is Python, more specifically, it has been used the 2.7 Python version.

1.2.2 Theano and others frameworks

Theano is the main framework used to develop the deep learning code. But others libraries has been used to, NumPy,Scikit-learn and matplotlib are the most relevant. In addition to this, other Python packages has been used to like Pickle.

Theano

Theano [1] is a Python library that allows users to work with mathematical expressions and work with simbolic variables, moreover, Theano handles multidimensional arrays efficiently. This framework has been used in order to build convolutional neural networks architecture and its training procedure.

There are numerous open-source deep-libraries that have been built on top of Theano, for example Keras, Lasagne and Blocks. Although the usability of using those libraries in stead of Theano is bigger, Theano is more flexible when user wants to develop its own

layer, for example.

Theano is not the unique language oriented to deep learning, for example Google, has developed its deep learning language called TensorFlow; Caffe and Torch are others examples.

Theano main page is available in http://deeplearning.net/software/theano/ where the documentation, examples,.. could be found.

NumPy, Scikit-learn and Matplotlib

NumPy is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. The documentation of this library is available in https://docs.scipy.org/doc/numpy/.

Scikit-learn is an open source Python library and commercially usable that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms which has been build in top of Numpy and matplotlib. Its documentation is available in the following url http://scikit-learn.org/dev/index.html.

Matplotlib is a python library for 2D plotting.

1.3 Databases

Some databases has been used in order to learn, compare results and carry out the project. All the databases are formed by three subsets whose samples are not repeated among subsets: train, validation and test.

- The training subset is used to train the network during epochs. To know how the training behavior, a cost is calculated.

- The validation subset is used to check the behavior of the network while is training, also, the validation subset is usually used to calculate the hyper-parameters of the network, although the hyper-parameters are not calculated until is pointed. The validation error is calculated for each training epoch. The metric use to the validation is $error(\%) = cost * 100$.

– The test subset, that is used just at the end of the training. The best model is chosen with regard to the best validation error. Different metrics are going to be used for after testing the network (error (%), TP, FP ...).

1.3.1 MNIST digit database

MNIST digit database is a image database of human written digits. This database is commonly used to learn machine learning techniques. Because of that, this database has been used in order to learn Theano and convolutional neural networks. In addition, this database has been used in a implemented convolutional neural network (LeNet).

Some examples of the digit image MNIST database could be seen in 1.1 and the charac-

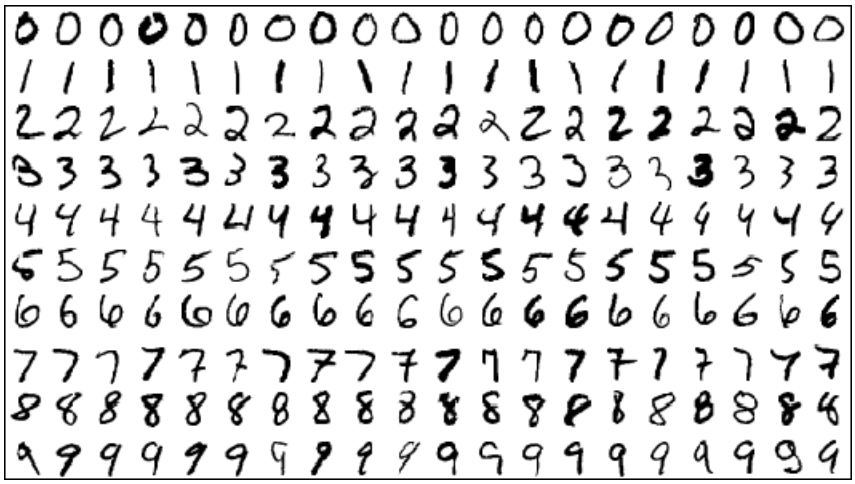


Figure 1.1: digits MNIST images database

teristics of this database are the following ones:

– Number of samples: 70.000 number of unique samples.

– Number of features/ length of each image: 784

– Number of classes: 10, one per digit

– The size of each image is 28x28 pixels.

– The images are in grey scale.

1.3.2 Labeled faces in the wild

The Labeled Faces in the Wild is a well-known and used dataset of faces images that can be found in its official web page http://vis-www.cs.umass.edu/lfw/.

The characteristics of this database are the following ones:

- Number of samples/ Number of images : 13233
- Number of features/ length of each vectorized image: 187500
- Number of classes / Number of people: 5748
- The number of images per person is not the same for each one.
- The size of each image is 250x250 pixels.
- The images are RGB ones.
- The faces in images are in the center of the image.

Figure 1.2: Examples of Labeled faces in the wild imamges

Examples of images of this database could be seen in figure 1.2 in which faces of well-known people are visualized.

This database has been used to learn; to learn how to read a database, how to feed the network with those images.

1.3.3 FRAV dataset

FRAV database is an anti-spoofing face database built in the research group of tu URJC called FRAV. This dataset is formed by five different classes:

- Original images of people.
- Images of people printed (attack).
- Images of people with a mask (attack).
- Images of pleople with a mask with the eyes cropped (attack).
- Images of people in a tablet (attack).

There are the same number of samples in each class. The images of that classes can be found in RGB and NIR (not all RGB images has its corresponding NIR image). Characteristics of FRAV images ere the following ones:

- There are 939 people in each RGB class or 195 in NIR class.
- There is one image per person.
- Each image h as it is own shape.
- The faces in images are in the center of the image.

This images has been used in different ways, in some experiments, just RGB images has been used, so 939 images has been used. When has been used RGB and NIR images at the same time, 195 images has been used (195 RGB and its corresponding NIR). To classify, two different ways has been used; the first one where real people formed one class and the different attacks formed other class, so two classes have been used; and the second way where each attacks correspond with a class, so five classes (4 attacks and 1 real) have been used.

One exaple of RGB images of FRAV database are shown in figure 1.3 and another example could be seen in 1.4. In both images, the four attacks described previously and the real user could be visualized.

1.3.4 CASIA dataset

Casia is a face anti-spoofing database built in the center for Biometrics and security research.

In the same way as FRAV dataset, this database is formed by real or genuine images of people and three different attacks of the same people:

Figure 1.3: Four attacks and real user from RGB FRAV database

Figure 1.4: Four attacks and real user from RGB FRAV database

- Images of people printed (attack).
- Images of people with a mask (attack).
- Images of people with a mask with the eyes cropped (attack).
- There are 939 people in each RGB class or 195 in NIR class.
- There is one image per person.
- The size of each image is 720x1280 pixels.
- The faces in images are in the center of the image.
- Images are RGB.

Two people of Casia database could be seen in figure 1.6 and figure 1.5, where three attacks types could be seen with the real user of both examples.

Figure 1.5: Three attacks and real user from casia database

Figure 1.6: Three attacks and real user from casia database

1.3.5 MSU - MFSD database

The MSU Mobile Face Spoofing Database (MFSD) is a video face anti-spoofing database although for this work just images have been used.

The database consist on users and attacks of the same people:

– Genuine users

–

The characteristics of the database are the following ones:

– 35 images per attack or genuine user.

- Images are RGB.
- Faces are centered in images.
- The size of each image are not equal. Approximately images are 300px heigh and 335px width.

In figure 1.7 and figure 1.8 It is posible to see the three attacks and the real user.

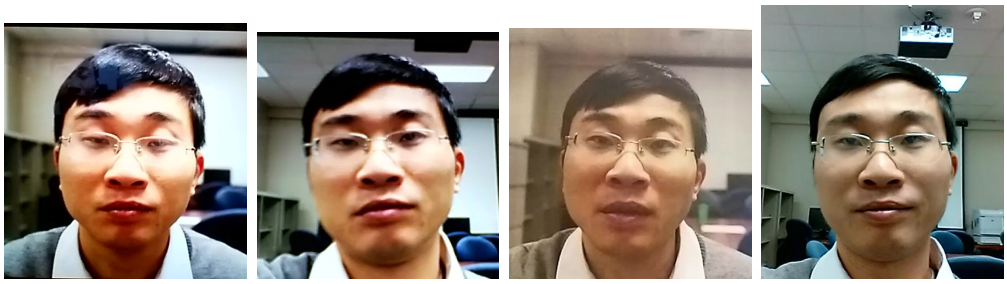


Figure 1.7: Three attacks and real user from a user of MFSD database

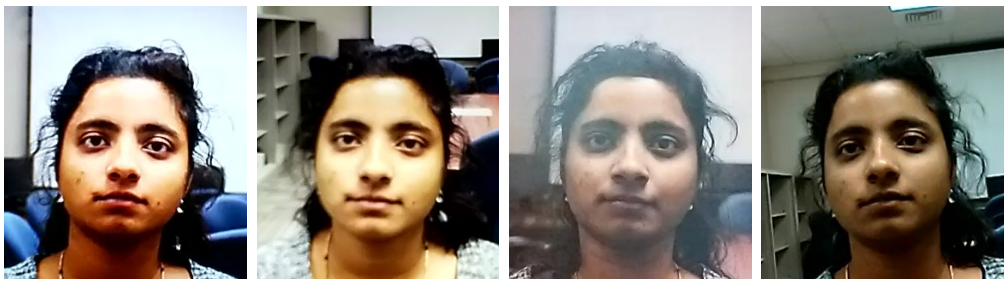


Figure 1.8: Three attacks and real user from a user of MFSD database

Chapter 2

Methodology

2.1 LeNet

LeNet is of a convolutional network designed for handwritten and machine-printed character recognition. The net has been proved with MINST database, a database of handwritten digits that comes with the net.

LeNet has been used to learn the implementation of convolutional neural networks, and LeNet is the basis of the network developed to carry out this project as it has been modified in order to adapt it to get this project goal.

The architecture of LeNet is formed by two convolutional layers of size 5x5 and with 20 kernels in the first convolutional layer and 50 in the second one, those are followed (each one) by a max pool layer of size 2x2. Those four layer are followed by a fully-connected layer with 500 neurons at the output. The classifier which has been used is the logistic regression. The activation function of the convolutional layers and the logistic regression is tanh.

The learning rate is 0,1 and the network runs by 200 epoch. In each epoch the training is realized with the training data set and the validation with its corresponding validation set of data, if the validation has its best performance then the test is realized.

The code of LeNet could be found in www.deeplearning.net, where the architecture, the training, test, validation, a early-stopping of the net, all the necessary to run the convolutional neural network is developed.

www.deeplearning.net is the original page of Theano where could be found tutorials, how to install and get it and the documentation of its functions.

2.1.1 Structure of Lenet

LeNet works basically with theano although it uses others libraries like NumPy.

First of all in Lenet the data is loaded. The function that download the data check, firstly, if the data is not downloaded yet, if it downloaded it does not repeat the download. The data is downloaded split into train, test and validation subsets.

After loading the data, the architecture is defined, the layers are called because they has been defined as objects, each kind of layer, one different object. There is a object for convolutional + maxpooling, another object for the hidden layers that is a fully connected layer, and the classifier used that is logistic regression. Each part of the code could be found in www.deeplearning.net

After creating the structure, the functions of training, validating and testing are created as Theano functions.

In a big while loop the training validation and testing is developed. The data is trained. Each mini-batch (that is a small quantity of that given by the user, it is used in order to not train or test all the data together because it would take too much computer resources) is trained and the weights and bias of each layer are updated. It is given a validation frequency, this parameters decides how many validations are produced. So the data is trained and validation, when a best score of validation is produced, the data is tested.

The training would run for a number of epoch that the user has given or by a early-Stopping that has been developed by authors. The early-stopping combats over-fitting by monitoring the model's performance on a validation set.

2.1.2 Results of LeNet

The result of the network using the given database could be seen in figure 2.1, where the least error in validation performance is 0.91 % obtained at iteration 18300, with an error test performance of 0.92 %.

It is important optimize the cost function, so it is possible to analyze how the networks works looking the cost function. The cost produced at the training gets reduced with the increasing iterations, as it is seen in figure 2.2.

As it is known, weights are filters, twenty first weights of each convolutional layers are shown in figure 2.3 of the tenth and hundredth epoch.

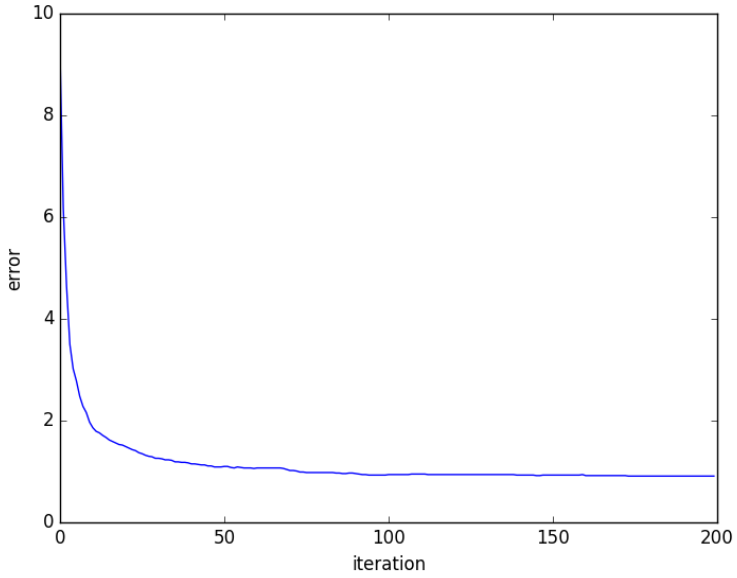


Figure 2.1: Validation error obtained with Lenet and MNIST digits database

While the network is learning, weights change and they would not have the same apparence if were represented.

Each time we take a sample and update our weights it is called a mini-batch. Each time we run through the entire database, it's called an epoch. Batch size determines how many examples you look at before making a weight update. The lower it is, the noisier the training signal is going to be, the higher it is, the longer it will take to compute the gradient for each step. http://stats.stackexchange.com/questions/140811/how-large-should-the-batch-size-be-for-stochastic-gradient-descent.

To be sure that the batch size do not affect to the results, it has been changed, in first place into 20 and in second place into 100. In the fist example (when batch size = 20), the while has been break because of early-stopped at epoch 31, from epoch 16 the networks does not improve the validation result.

In figure 2.4 the error in each epoch is represented for 500 batch size, the original size, for a value of 20 and 100. In the original case, the error stars with a value of 9% aprox. with the batch size = 20 the error in the first iteration is about 2.4%, and with a bunch of 100 images, the validation error is 3.5%.

With the original size and size equal to 20, it is possible to get to the same minimum, the difference between those examples is that each one get to that conclusion into different

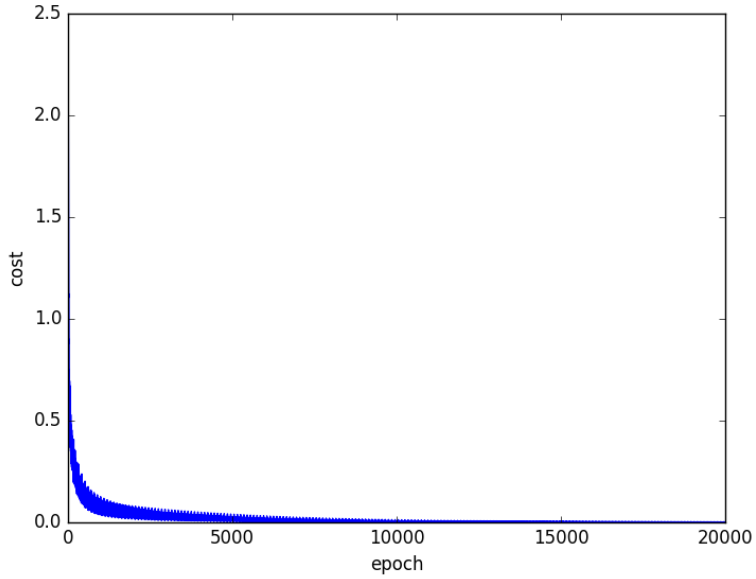


Figure 2.2: Cost function of running Lenet with it is own database.

epochs. With a batch size equal to 100, the code stopped because of the early-stop with a patience of 10000; it stop in epoch 33,while those epochs, it has been possible to get to a test error of 1.04% in iteration 8500 when the validation error was 1.01%, it has been running with put getting a better validation score for 17 epoch. With a batch size = 20, the code also has stopped earlier because of the same reason, but in that case it has been possible to get to the same minimum that with the original size; the epoch in which has stopped is 31, it has been running without getting a better validation score for 15 epochs.

If the patience is increased from 10000 to 100000 for a batch size = 20 and equal to 100, results obtained are that for a value of 20, it has been running for 40 epoch, having the same results that the one obtained with a patience of 10000.

For a value of 100 with the patience increased to 100000, the results are not the same as the one obtained the patience equal to 10000. In this occasion, the results are better than in the original one; The code has run for 258,08 minutes and for 200 epochs. The result obtained was better than in the original one has it has said. In epoch 51 has gotten a validation error of 1 % and the error test has been 0,89%, from this point, the test error that has been calculated five more times, has been increasing until get the value 0,85% in iteration 55500, epoch number 111, the validation error obtained has been 0,95%. From this epoch until number 200, the network has not had any better result.

In figure X it is possible to see the error function for those two examples. It could be

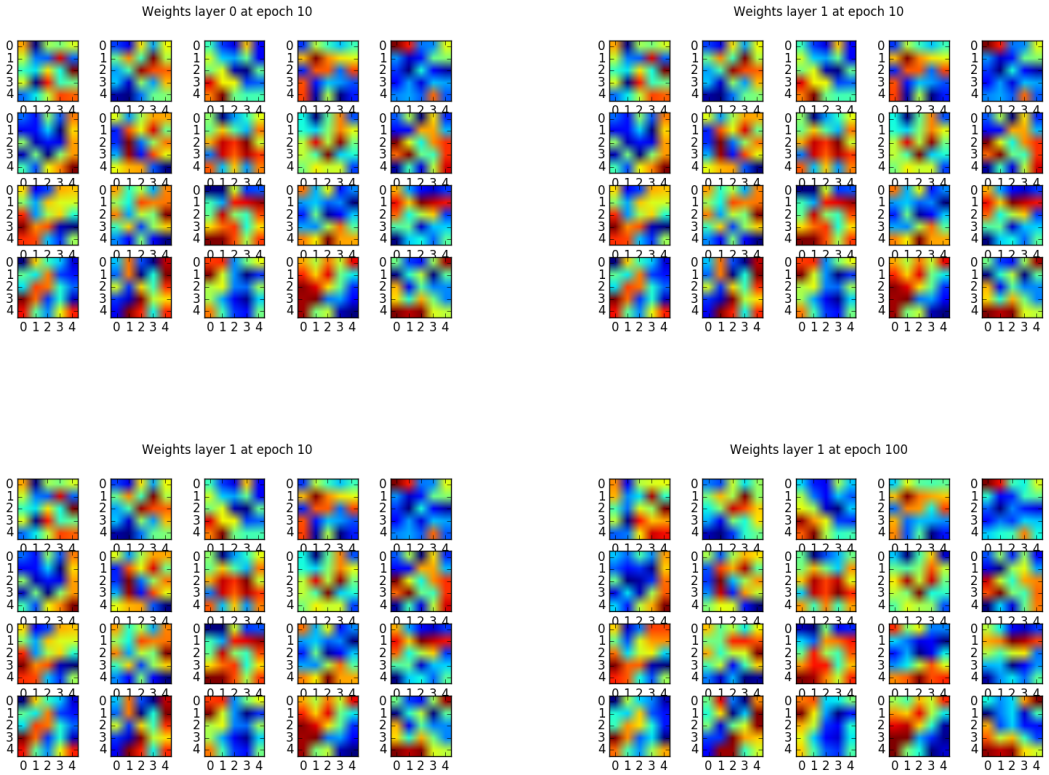
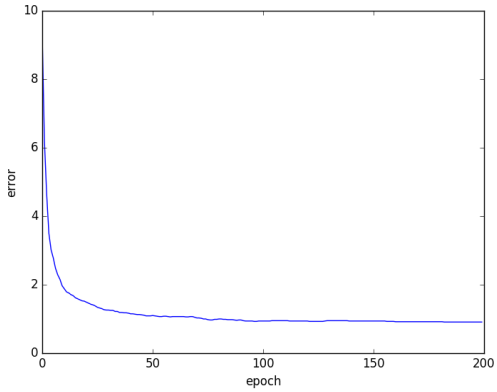
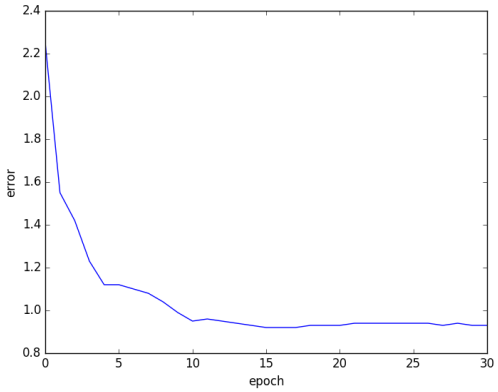


Figure 2.3: Weights at epoch 10 and 100 of the two convolutional layers

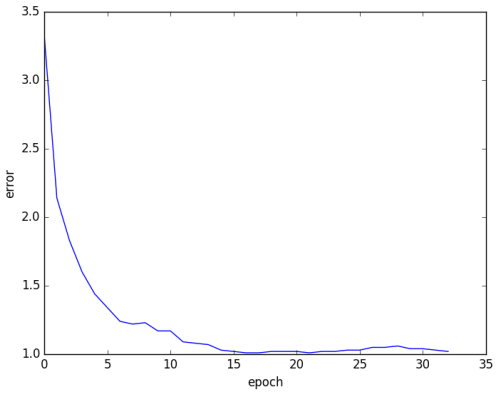
seen that the example with batch size = 20 has just run for 40 epoch and the good results of the example with batch size = 100, remembering that the patience has been increased from 10000 to 100000.



(a) batch size = 500 (original size)



(b) batch size = 20



(c) batch size = 100

Figure 2.4: Validation error in each epoch for diferentes sizes of batches.

2.1.3 Modifying LeNet

From LeNet, with the database gray scale digit database, the one that LeNet uses in the original example (grey scale images whose size is 28x28), some modification has been realized:

- Using Local Response Normalization (LRN) In which a normalization has been carried out in the convolutional-max pooling layers.
- Using ReLu as activation function: The activation function tanh has been substituted by ReLu activation function in convolutional and fully connected layers.
- Using Relu and LRN: The activation function used is ReLu and LRN has been used as normalization layer.
- Change weights initialization: Weights initialization has been changed by Gaussian. In which mean value that has been used is 0 and std is 0.01. Weights initialization has been changed in convolutional and fully connected layers. Also, bias initialization has been changed by ones.

First, the cost of training process are going to be visualized with the original cost train, without modifying LeNet). In figure 2.5 is represented. Also the validation error (validation cost*100) could be visualized in figure 2.5. The network has been running for 200 epochs.

Visualizing the loss during the training, could be affirmed that the network with Gaussian initialization is in a local minimum because the loss has converged as could be seen in figure 2.5(e). The loss of LeNet without being modified (figure 2.5(a)) is the one whose oscillation at training is less than others.

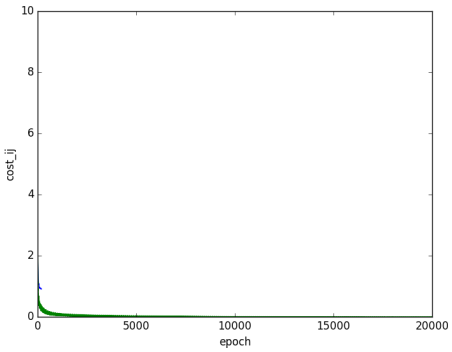
About the error at validation, visualizing the graphs in figure 2.5, it is very similar the curve for original lenet, lenet with ReLu, lenet with LRN and lenet with LRN and ReLu.

At testing, the results obtained have been the following ones:

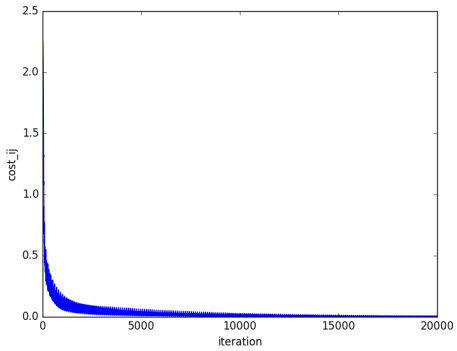
- Original LeNet: Best validation score of 0.91 % obtained at iteration 17400, with test performance 0.92%.
- Using Local Response Normalization: Best validation score of 0.99 % obtained at iteration 13400, with test performance 1.6 %.
- Using ReLu as activation function:Best validation score of 1.04 % obtained at iteration 11900, with test performance 2.4%.

- Using Relu and LRN: Best validation score of 1.18 % obtained at iteration 19500, with test performance 1.08 %.
- Gaussian weight initialization: Best validation score of 81.22% obtained at iteration 100, with test performance 80.90%.

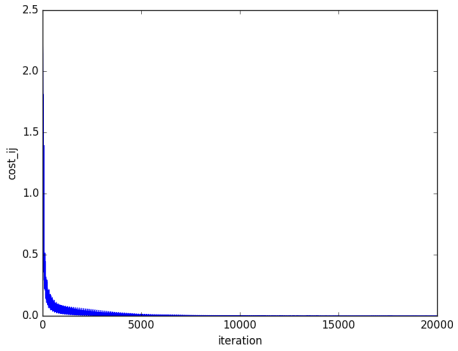
The best configuration for the network is the original one. With Gaussian initalization, the network does not find a local minimum in such a sort of time. Using LRN and Relu, test result is closer to the obtained with LeNet original, but not as good as the last one. Changing the activation function has not been a good change. Not taking into account original LeNet, the best test performance has been obtained with 1,08% using ReLu and LRN, but the best validation error is 0,99% obtained using just LRN. The values are close of the modifications, but the modification of Gaussian weight initialization.



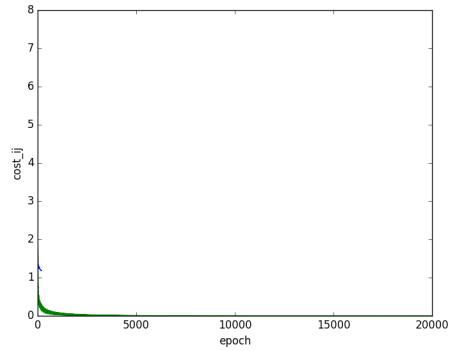
(a) Original LeNet



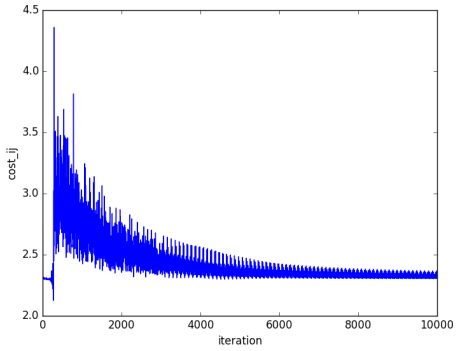
(b) Using LRN



(c) Using ReLu

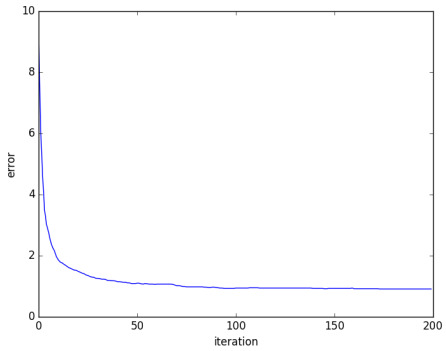


(d) Using ReLu and LRN

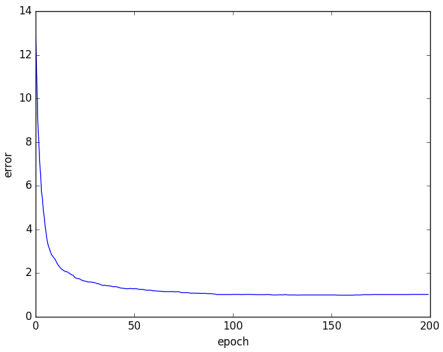


(e) Using Gaussian weights init.

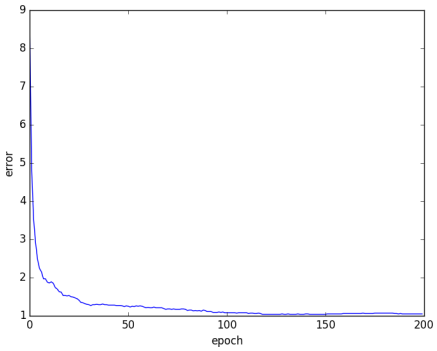
Figure 2.5: Cost function of Lenet and Lenet Modified.



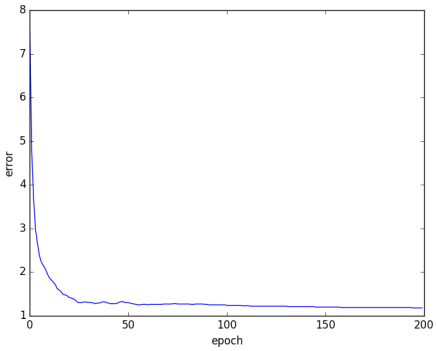
(a) Original LeNet



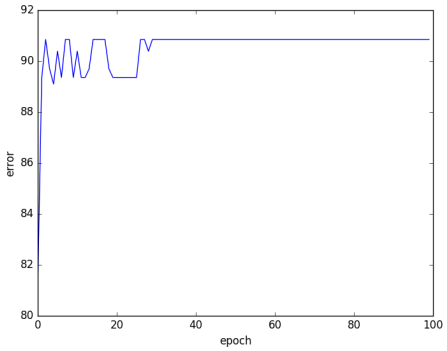
(b) Using LRN



(c) Using ReLu



(d) Using ReLu and LRN



(e) Using Gaussian weights init.

Figure 2.6: Valid error of Lenet and Lenet Modified.

Bibliography

[1] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

List of Figures

1.1 digits MNIST images database 5
1.2 Examples of Labeled faces in the wild imamges 6
1.3 Four attacks and real user from RGB FRAV database 8
1.4 Four attacks and real user from RGB FRAV database 8
1.5 Three attacks and real user from casia database 9
1.6 Three attacks and real user from casia database 9
1.7 Three attacks and real user from a user of MFSD database 10
1.8 Three attacks and real user from a user of MFSD database 10

2.1 Validation error obtained with Lenet and MNIST digits database 13
2.2 Cost function of running Lenet with it is own database. 14
2.3 Weights at epoch 10 and 100 of the two convolutional layers 15
2.4 Validation error in each epoch for diferentes sizes of batches. 16
2.5 Cost function of Lenet and Lenet Modified. 19
2.6 Valid error of Lenet and Lenet Modified. 20

List of Tables