Practical work N°4

(PROJECT)

General remarks

The purpose of this project is to become familiar with the use of neural networks to solve computer vision tasks. You will handle several types of neural networks as well as several common datasets. The project has four parts:

1. The use of a neural network to classify the handwritten figures present on an image;
2. The implementation of a random image generator and the use of a second neural network capable of indicating whether an image includes a handwritten figure or not;
3. The implementation of a neural network capable of generating images seeming like handwritten figures;
4. Using the skills acquired in the three steps above to classify other types of images.

During the project, you are strongly encouraged to search for information by yourself about the network architectures proposed by others. Feel free to take inspiration from existing publications or tutorials (by citing them). Some useful links on neural networks:

- Notes from Stanford University: http://cs231n.github.io/
- The illustrated backpropagation: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

To conduct this project, you should be organized on groups of 2 or 3 three students.

At the end of the project you will have to submit a report on Moodle of your activity showing your developments as well as your illustrated results.

Part 1: MNIST Dataset

MNIST is a subset of the NIST database (National Institute of Standards and Technology). It was developed by the precursors of deep learning, Y. LeCun and Y. Bengio, in 1998. It contains manual data writing data from 0 to 9. It generally leads to a multi-class classification problem to 10 classes. In its initial form, the training sample comprises 60,000 examples, and 10,000 in test. An input example is a 28 × 28 fixed size image, each pixel being white (0) or black (1). The small size of images (28 × 28) makes it possible to quickly train neural networks.

It is possible to load MNIST under ***sklearn*** with the instruction ***mns = fetch_mldata ("original MNIST")*** of the datasets package. Then, the learnings will be done on the first 60000 examples, and the tests will be done on the remaining examples in the loaded sample. This is the protocol used by all researchers testing their algorithms on MNIST.

Part 2: Handwritten digits images classification

In this part, you will need to create a neural network that takes a grayscale image from the 28 × 28 size MNIST dataset and returns the label of the digit on the image. For this purpose, you need to study successively the influence of:

- Number of iterations;
- Learning step value

Moreover, the comparison could be performed according to:

- The evolution of the cost on the bases of learning and test;
- The recognition rate obtained using the WTA decision rule (Winner Takes All: the most active output cell gives the winning class).

So, let start!

Question 1:

1. Display a couple of examples;
2. Propose and train a neural network on the MNIST dataset (training examples only);
3. Visualize the evolution of the model precision during learning (see training history)
4. Visualize the prediction of the neuron network for an input image:
   - View the input image
   - Visualize the predictions of the network (probabilities of belonging to the classes)
5. Evaluate classification rates in learning on the whole MNIST test set.
6. Change the distribution of the examples in the following way: 60% in the learning base and the rest in the test base. Optimize the structure of the neural network (number of cells in hidden layer);
7. Study the influence of the number of iterations;

Question 2: Cross-validation

In order to improve performance in the generalization of the neural network, it is proposed to implement a learning with "cross-validation".

1. Distribute MNIST data in learning base (60%), cross-validation (20%) and test (20%);
2. Train a neural network with "cross-validation";
3. Conclude.

Question 3: Probabilities estimation

Write a program allowing the estimation of the a priori probabilities of each of the 10 classes of digits from the calculation of the frequencies of appearance of the classes on all the base. Display these probabilities.

Question 4: Probabilities estimation with neural network

1. Calculate the outputs (probability a posteriori) of the network for all the examples of the base MNIST;
2. For each case, check whether the sum is equal to 1 or not.

Question 5: Classification and rejection

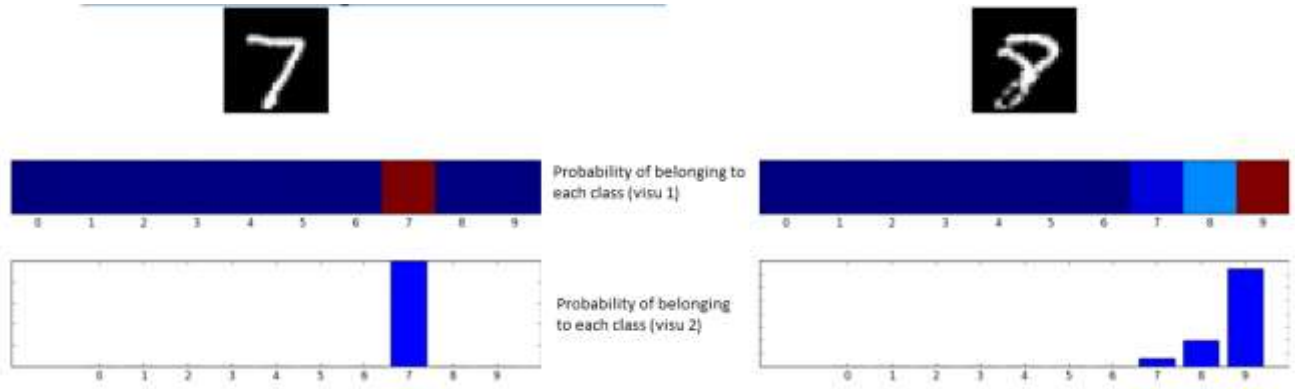
1. Calculate classifier recognition rate on the entire base;
2. It is proposed to improve this score by allowing rejection in the decision stage (see both types of rejection seen in lecture 2).
   - Calculate for each type of rejection the scores obtained for a dozen values of the rejection thresholds. You must provide the rejected sample rate for each of these values;
   - For the two types of rejection, plot the rejection rate curves as a function of the threshold value and the recognition rate curve also as a function of the threshold value;
   - Interpret these curves;
   - Conclude.

Part 3: Detect the presence of a digit on an image

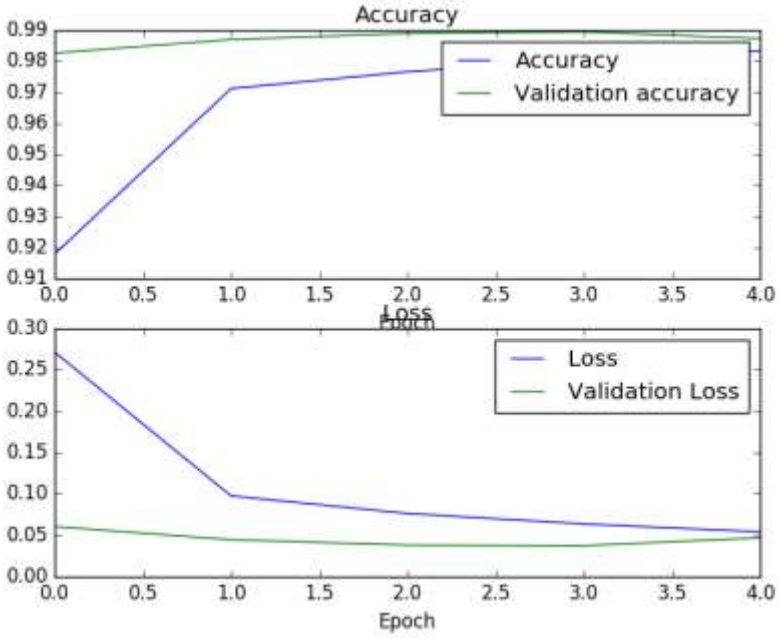
In this part, you will create a neural network capable of identifying whether a 28 × 28 gray-scale image has a digit or not. So, it will be necessary to use images outside MNIST to drive the network. For this you will need:

1. Generate "random" grayscale images of size 28 × 28 according to one of the following strategies:
   - Create a python function that generates an image in which pixel values are derived from a probability distribution;
   - Randomly collect patch size 28 × 28 in other images.
2. Propose an architecture for the classifier network (digit / non-digit)
3. The classifier network will be trained on 30,000 images randomly selected from MNIST and 30,000 images generated randomly:
   - Evaluate the classifier's performance using the set of 10,000 MNIST test images with 10,000 randomly generated images;
   - Propose a visualization of the prediction of the classifier.

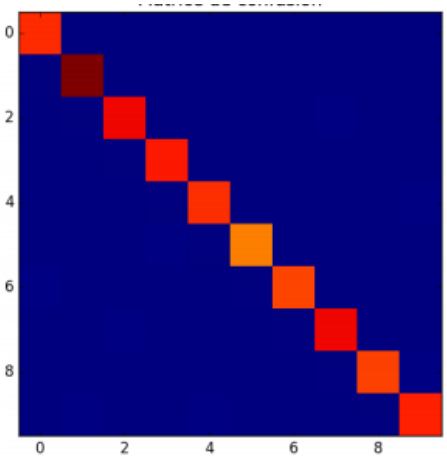
Example 1: visualization of network predictions



Example 2: visualization of the performances of the network during its training



Example 3: confusion matrix



Part 4: Generate realistic images of digits

In this part you will create a neural network capable of generating images resembling digits present in the MNIST dataset. You will follow the GAN (Generative Adversarial Network) approach described as following:

A GAN is constituted by two networks of different goals:

- A *Generator* network that takes input from a number of variables and produces an output image (grayscale and 28 × 28 in our case);
- A *Discriminant* network that takes an image (grayscale and size 28 × 28 in our case) as input and predicts if the image represents the object of interest (a digit in our case).

A GAN is a network in which we have paired these two parts (the output of the *Generator* is sent to the *Discriminant*). In a GAN, the *Discriminant* and the *Generator* counter iteratively to improve and thus cause the *Generator* to produce the desired result.

Some useful links on GAN:

- https://en.wikipedia.org/wiki/Generative_adversarial_network
- https://arxiv.org/abs/1406.2661
- http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

So, in this part, the objective is:

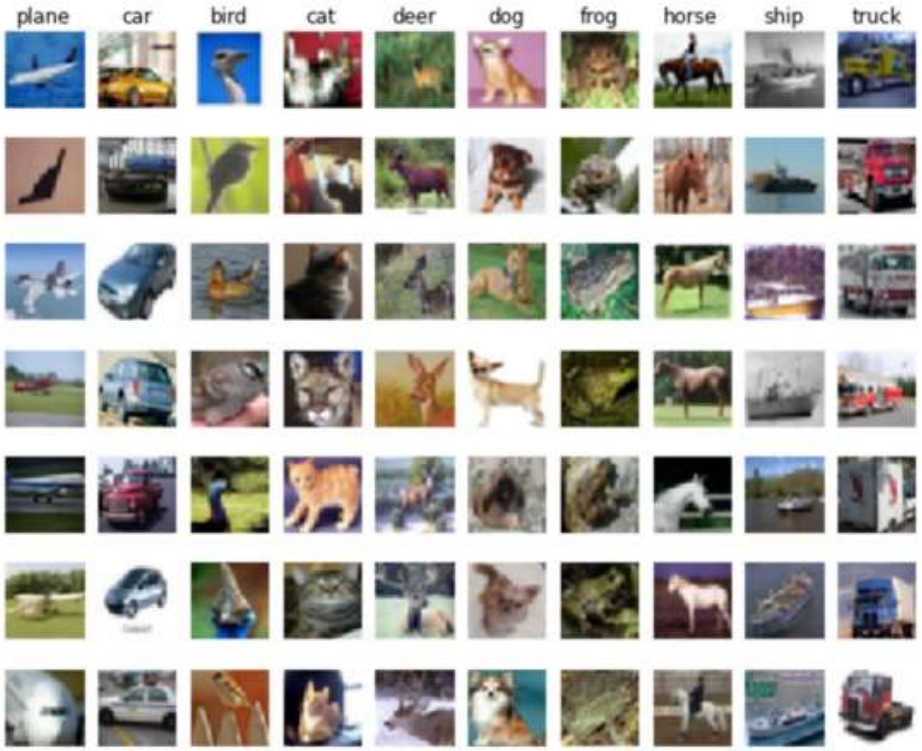
1. Propose an architecture for the "Discriminant" network ;
2. Propose an architecture for the "Generator" network ;
3. Pair the two networks to form the GAN;
4. Describe the GAN training procedure;
5. Train the GAN;
6. At the end of each learning episode of the GAN (or after a number of iterations of Batch trainings), illustrate a set of images generated by the network.

Part 5: Generate realistic images of digits

After having mastered the concepts and implementation of neural networks, this fourth step of the project allows you to give free rein to your desires and your imagination! You will need to use the knowledge you acquired in the previous three steps to perform classifications on more complex datasets than MNIST.

For example, use images from **CIFAR10** to classify color images (be careful of the shape of the input data!), or images from the **MNIST Fashion** dataset that allow you to push your developments on grayscale images with more complex content.

Example 1: CIFAR10



Example 1: MNIST Fashion