Neural Networks: Multi-Layer Perceptron and Convolutional Neural Networks

E. Gaussier, O. Goga, P. Loiseau (M. Tami, T. Thonet)

20 March 2020

This lab contains two parts: first a part on MLPs with the sklearn tool that you have used during the previous lab; and second a tutorial to use the more powerful tool PyTorch with the example of building a CNN for image classification.

Note: Due to the university closure, you will be doing the lab on your own at home. We will be available to assist online at the regular lab time, Friday 8am–1:15am. We also ask that you do the lab in one (or several) Jupyter notebooks where you both put your code and answer questions, and that you send us the notebooks by email at `eric.gaussier@imag.fr` and `patrick.loiseau@inria.fr`, by Sunday midnight.

Part I: Multi-Layer Perceptron with sklearn

The goal of this work is to study neural networks and in particular one of their common building blocks, namely *Multi-Layer Perceptrons* (MLPs). To this end, we will use the function **MLPClassifier** from the module **sklearn.neural_network**. This function, that provides a generic implementation of MLPs for classification purposes, is based on several parameters such as the number of neurons on each hidden layer (**hidden_layer_sizes**), the activation/squashing function (**activation**) and the optimization algorithm (**solver**). Do not hesitate to look at the online documentation available at:
scikit-learn.org/stable/modules/generated/sklearn. neural_network.MLPClassifier.html

1 Learning Boolean Operators

In this first exercice, we want to learn the boolean operators AND, OR and XOR with an MLP. These operators are defined by:

In the remainder, we will use a matrix $\mathbf{X}$ to represent the possible inputs and a vector $\mathbf{y}$ for the associated outputs. For example, for the AND operator:

```
X = [ [0., 0.], [0., 1.], [1., 0.], [1., 1.] ] # Inputs
y = [0, 0, 0, 1] # Outputs
```

Input		Output		
X_1	X_2	X_1 AND X_2	X_1 OR X_2	X_1 XOR X_2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

After having specified a classifier from the **MLPClassifier** function, it can be trained on couples $(\mathbf{X}, \mathbf{y})$ by calling **classifier.fit(X, y)**. This classifier can then be used to predict the outputs associated with given inputs **X_test** by calling **classifier.predict(X_test)** (note that **X_test** can be equal to **X**).

1. Define an MLP classifier to learn the AND operator. This classifier should have no hidden layers, a linear activation function (**identity** function) and rely on the **lbfgs** solver. Check that the results predicted are correct.
2. Define an MLP classifier to learn the OR operator. As before, this classifier should have no hidden layers, a linear activation function and rely on the **lbfgs** solver. Check that the results predicted are correct.
3. Define an MLP classifier to learn the XOR operator:
   (a) Using no hidden layers, a linear activation function and the **lbfgs** solver. Are the predicted results correct? How do you explain that?
   (b) Using two hidden layers comprising 4 neurons (first layer) and 2 neurons (second layer), linear activation functions and the **lbfgs** solver. Are the predicted results in this case correct? How do you explain that?
   (c) Using two hidden layers comprising 4 neurons (first layer) and 2 neurons (second layer), non-linear activation functions as the hyperbolic tangent function (**tanh**) and the **lbfgs** solver. Retrain the model several times. Are the results predicted correct? How do you explain that?

2 Image Classification

We are now interested in a more complex learning problem: identifying hand-written digits in images. The dataset to be used is **digits** from **scikit-learn**. 90% of this dataset will be used for training, and 10% for testing:

```
from sklearn.datasets import load_digits dataset = load_digits ()
X = dataset.data # Inputs
y = dataset.target # Associated outputs
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=0.1)
```

Learn classifiers on **(train_X, train_y)** playing with the MLP parameters: number of hidden layers, number of neurons on each hidden layer, activation functions, solver, ... The performance of each classifier will be evaluated on the test set on the basis of the classification accuracy:

```
from sklearn.metrics import accuracy_score
test_y_pred = classifier.predict(test_X) # Predicted results
print("Accuracy :",accuracy_score(test_y,test_y_pred))
```

Which classifier performs best? Comment the results you obtain.

Part II: PyTorch and convolutional neural nets

The second part of the lab consists in implementing a CNN using PyTorch.

First, you will go through the standard PyTorch tutorial available at `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`. You will need to install `torch` and `torchvision`. See `https://github.com/pytorch/pytorch#installation` for instructions. The easiest in general is to install directly from binaries, see `https://pytorch.org/`. Then, go through each of the four parts of the tutorial:

- `https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html`: basics of tensors (ignore the last part on CUDA tensors)
- `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html`: autograd for automatic differentiation
- `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`: basic neural nets
- `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`: Cifar10 classifier with CNN (ignore the last part on training on GPUs).

For each of the four parts, you can either download the corresponding notebook or execute it on Google colab. Execute all the different pieces of codes and play with them to understand what is behind the different functions.

Second, in a new notebook of your own (or in the same as the one where you did the MLP in Part I), do the following:

- Implement the LeNet network as in the fourth part of the tutorial above for the MNIST dataset (from `torchvision`).
- Compute the accuracy and compare it to the MLP accuracy; comment.
- Modify the network architecture (size of feature maps, size of kernel); comment.
- Modify the learning rate; comment.
- Bonus: implement dropout regularization in the training.

For each point, please include code and comments below it.