Practical Session : Gradient Descent

Rémi Flamary

During the practical session you will need to use the Python `numpy` `pylab` and `scipy` and `autograd` toolboxes. You can install `autograd` on conda with the following command: `conda install -c conda-forge autograd`

It is recommended that you import them at the beginning of all your scripts with the following code :

```
import autograd.numpy as np
import pylab as pl
import autograd.scipy as sp
import autograd
```

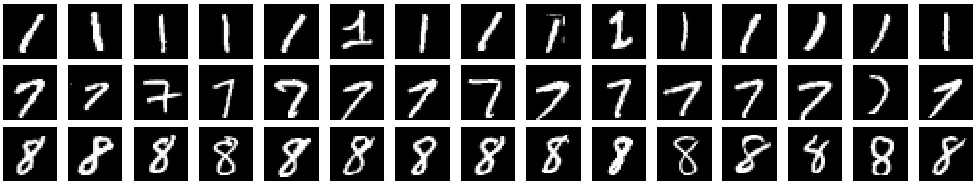
Note that using `autograd.numpy` means that you are using numpy but the autograd toolbox can store operations and perform automatic differentiation

1 Dataset and gradient

1.1 Dataset and illustration

- Download data file "digits.npz".
- Load the file in memory using function `np.load`. The file contains the following matrices:
  - `x` and `xt`: data matrix containing respectively $n = 3000$ and $nt = 1500$ trining example of manuscript digits. Each line in those matrices is a 28×28 image stored as a transposed vector (a line).

    Some examples of the images in the training sets:

  - `y` and `yt`: the labels of the images described above. they are vectors containing the classes $(1, 7, 8)$ of each images in `x` and `xt`.
- Use function `np.reshape` to reconstruct 28×28 images. Visualize a few examples from each class using function `pl.imshow`.
- Normalize the data and use the normalize data in the following (`np.mean`,`np.std`). Be careful not to introduce `NaN` values.
- We want to create a binary classification problem from those 3 classes. You can first try and discriminante class 8 VS 1 and 7. Compute a vector `yb` containing labels $(-1, 1)$ for training and `ytb` for testing (operator `==`) .

1.2 Cost and gradient

- Compute the training matrix $\mathbf{X}$ by adding a column of 1 to estimate the bias.
- Code a function `cost` that compute the following cost for the regularized logistic regression with labels $\{-1, 1\}$ and $reg = 1$:

$$cost(\alpha) = \sum_{i=1}^{n} \log(1 + \exp(-y_i (x_i)^T \alpha)) + reg * \sum_{j=1}^{d} \alpha_i^2$$

  Note that the x_i^T are the lines from matrix $\mathbf{X}$, α is of dimensionality $d + 1$ and α_{d+1} is the bias (constant term). Test the cost function for a vector full of zeros.
- Use the function `autograd.grad` on the cost function to compute the gradient of the cost. Test the gradient for a vector full of zeros and check that it provides a descent direction.
- Use the function `autograd.hessian` on the cost function tu compute the hessian of the cost. Compute the Hessian matrix for a vector full of zeros. Is it positive definite?

Note that while automatic differentiation is provided by autograd and all major neural network toolboxes (pytorch,tensorflow), the gradients and especially the Hessian can be computed on this problem more efficiently with a few matrix product. Don't always use autodiff when the function is simple!

2 Gradient descent

In this section you will implement some gradient descent methods and also use generic solvers from `scipy.optimize`.

2.1 Gradient descent

- Code the gradient descent algorithm using the gradient function with a fixed step for 1000 iterations. Store the cost along the iterations in a python list.
- Plot the evolution of the cost. Is there convergence? compute the norm of the gradient at the last iteration.
- Compute the classification accuracy for the binary classification problem.

2.2 Newton descent

- Code the gradient descent algorithm using the gradient function with a fixed step for 10 iterations. Store the cost along the iterations in a python list.
- Compare the convergence speed of the gradient descent and Newton. Compute the norm of the gradient at the last Newton iteration.
- Compare the computational time of Newton vs Gradient Descent.

2.3 BFGS/L-BFGS

- Use the function `scipy.optimize.minimize` to solve the optimization problem. The default method is the BFGS.
- Change the method to L-BFGS-B and compare the computational time.