Practical work 05 – 28/10/2019
Back-Propagation

Objectives

Main objective is to implement the back propagation algorithm for an arbitrary (fully connected) MLP with Softmax.

A suitable structure is again provided in form of a Jupyter Notebook which will guide you through implementing suitable python classes that will allow to easily specify MLP models of arbitrary depth and layer sizes.

The implementation should then be used to train MNIST with one or more hidden layers.

Submission

— **Deadline** : Wednesday 28 October, 22pm

— **Format** :

  — Completed Jupyter notebook with the blanks filled in (the sections to be completed marked as usual).

  — Comments and results (plot with learning curve showing the results for different model complexities) either in the notebook or in a pdf-report.

Exercice 1 Object-Oriented Python

Get up to speed with object-oriented python (if not up-to-speed already). See a suitable online tutorial (such as e.g. https ://realpython.com/python3-object-oriented-programming/).

Exercice 2 Implement Forward Propagation

Implement the forward propagation through an arbitrary MLP. Use the Jupyter notebook `backprop-stud.ipynb`. Complete the implementation of the method

— propagate

in the 'Layer' and the 'MLP' classes. Keep an eye on the shapes of the arrays defined as inputs, outputs or intermediate variables.

Check your implementation with the test following 'Check the Implementation of Forward Propagation'.

Then, measure the performance of your implementation by testing different mini-batch sizes (1-60'000) for forward propagating once the whole MNIST training set. Use the code right after 'Test Performance of Forward Propagation'. The notebook contains the 'Dataset'-class that can be used for loading the data and providing mini-batches.

Describe and interpret in a few sentences what you observe in the performance test.

Exercice 3 Implement Backpropagation

Now, implement backpropagation. Complete the implementation of the methods

— backpropagate

— gradient_weights

— gradient_bias

— update_params

(see the classes Layer, Softmax, MLP). Again, keep an eye on properly and consistently specifying the shapes of the arrays. Use suitable assert statements to check the shapes (some examples are given in the notebook e.g. in the 'backpropagate' method of 'Layer').

Check the proper implementation of the gradient of the MLP by running the gradient checking (after 'Check the Implementation of the Gradient'). This iterates through all the parameters, computes the numeric approximation and tests for discrepancies larger than a given accuracy ($3.0e^{-7}$). Numeric output is provided if this threshold is exceeded. Inspect how the numeric approximation of the gradient is computed.

Exercice 4 Train MNIST

Now, use your implementation to instantiate and train an MLP for MNIST with mini-batch gradient descent.

Study two different architectures :

a) Shallow Network : Single hidden layer layer with 150 units.

b) Deeper Network : Four hidden layers with 150, 200, 150, 50 hidden layers.

Fill the blanks in the cell after 'Training, Evaluating Performance'.

What are suitable learning rates, batch sizes, number of epochs ? Describe your findings including the achieved test error rates and the learning curves.

Exercice 5 Optional : Review Questions

a) Determine the number of model parameters of an MLP for original MNIST with 3 hidden layers of 100, 200, 50 units.