Teachers: J. Hennebert, M. Melchior
Assistants: C. Gisler, M. Willi, Y.-I. Beffa

Practical Work 05 – 24/03/2021
Back-Propagation

Objectives

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

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

Once implemented you will use it to train MNIST with one or more hidden layers and explore its capabilities.

Submission

— **Deadline** : Wednesday 7 April, 10am

— **Format** :

Completed Jupyter notebook with the blanks filled in (the sections to be completed marked as usual). Plots with learning curves showing the results for different model complexities, final performance achieved (in numeric format) **and comments** also in the same notebook.

Indicate the name of the group in the name of the notebook.

— **One submission per group !**

Exercise 1 Optional : 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/).

Exercise 2 Forward Propagation

Implement the forward propagation through an arbitrary MLP. Use the Jupyter notebook `backprop_stud.ipynb`.

a) *Implement* : 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.

b) *Check* : Check your implementation with the test following 'Check the Implementation of Forward Propagation'.

c) *Runtime Performance* : Measure the performance of your implementation by forward propagating once the samples of the training set chunked into different mini-batch sizes (selected values between 1 and the size of the 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. At what mini-batch size is the forward propagation fastest. Explain why it takes different amounts of time to forward prop all the training sample when using different batch sizes.

Exercise 3 Back-Propagation

Implement back-propagation.

a) *Implement* : 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*).

b) *Check* : Check 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 - gradient checking might be helpful for debugging your application !

Exercise 4 Train MNIST

Use your implementation to 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'. Find suitable learning rates, batch sizes, number of epochs for the two architectures. Describe your findings including the achieved test error rates and the learning curves.

Now try to beat the best error rate achieved by trying different number of layers and number of units per layer.

Exercise 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.
b) What is understood as 'curse of dimensionality' and in what sense is deep learning believed to have an answer to that ?
c) Describe your intuition on what the successive layers in a deep learning model learn during training and explain why this beneficial.
d) In what sense is the backprop algorithm more efficient than just a brute force calculation of the gradient ?
e) What are the variables in MLP that need to be remembered per layer during the forward pass to have them available in the backward pass ?