AI project: A Neural Network implementation

In this project you will implement a simple two-layer Neural Network, along with its training algorithm based on back-propagation. This project has two parts:

1. Implementation of a fully-connected Neural Network (NN) by using only basic matrix operations,
2. Implementation of NN by using pytorch functions.

This NN will be employed to classify the 10 object classes present in the popular CIFAR-10 dataset, that you will use as a benchmark. This dataset consists of 50k training colour images of 32x32 resolution with 10 object classes, namely aeroplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and lorries.

1. Neural Network implementation from scratch

In this first section you will be testing your results with the file `cw_nn.py`, which invokes the functions of your NN and compares its results with the expected ones. Your code implementing the architecture of the NN should be written in `simple_nn.py` in the indicated sections.

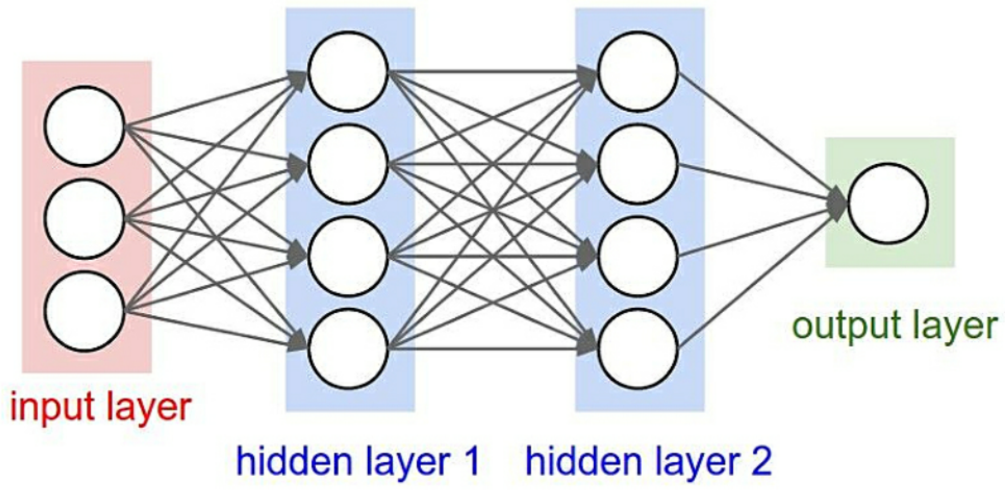

Figure 1: Architecture of a fully-connected Neural Network

The architecture of the NN is shown in figure 1. It has an input layer, two hidden layers and an output layer. Before training the model with CIFAR-10, you might want first implement all the steps using random data to facilitate debugging your code.

In this first stage of the implementation the NN receives an input $X \in \mathbb{R}^4$, which is directed to the first layer of the NN, with 10 neurons, via a linear weighting matrix $W_1 \in \mathbb{R}^{10x4}$ and the bias term $b_1 \in \mathbb{R}^{10}$. In this layer a linear operation is performed,

$$z_2 = W_1 X + b_1 \tag{1}$$

resulting in a 10-dimensional vector z_2. This operation is followed by a ReLU non-linear activation ϕ, applied element-wise on each neuron, which yields the activations $a_2 = \phi(z_2)$ according to the definition of ϕ:

$$\phi(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2}$$

The result a_2 is then processed by the second hidden layer, composed of 10 neurons as well. Here a_2 is multiplied by $W_2 \in \mathbb{R}^{3x10}$ and the bias term $b_2 \in \mathbb{R}^3$. In this layer the operation performed is

$$z_3 = W_2 a_2 + b_2 \tag{3}$$

Finally, the output of this layer z_3 goes through the output layer, where a sigmoid activation ψ function is applied element-wise, yielding the final output of the NN:

$$\psi(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{4}$$

The goal of this project is to develop the implementation of all the steps for this architecture. Specifically:

- Implement the feed-forward model of a two-layered NN architecture as well as the loss function to train it.
- Implement the back-propagation algorithm for this NN
- Train the model for toy random data using SGD
- Train the model for real data from the CIFAR-10 dataset using SGD and learning rate decay schedules.
- Improve the model training with better hyperparameters
- Optionally, add new techniques to further improve the performance of the NN.

2. Neural Network implementation with pytorch

In this section you will make use of the pytorch library to implement the same NN as before. You will then add more layers and adjustments to the hyperparameters to improve the performance of the NN.

Specifically:

- Implement the forward-pass, loss, gradient computation and optimisation steps with pytorch functions.
- Add more layers to the NN (up to 5) and tune the hyperparameters to improve its performance.

An important aspect of this work will be to make the most of python vectorisation in your code.

At the end of this week you will give a presentation in which you explain your choices when training and optimising the NN. This presentation may also include the relevant plots of the loss, training and validation performance, etc, as well as a discussion of your results.