Lab 3 Frequently Asked Questions

Q1: Why do we set the gradient to zero at each iteration?
A1: The reason is that when you update the weights at each step, using optimizer.step() you are adding the current gradient to one computed in the previous step (this is useful in some applications). Therefore to avoid the gradient to keep growing with each step, we set its previous value to zero.

Q2: Which amongst the change we made at the end of the lab is the most impactful on the performance?
A2: Using Adam instead of the Stochastic Gradient Descent (SGD). SGD, while very simple to understand and useful in many scenarios, fails to train more complex networks due to the presence of too many local minima. Moreover, SGD is famously known to have poor numerical performance in “ill-conditioned problems”.

Q3: What is the advantage of switching to the relu function?
A3: The derivative of the sigmoid function is very close to zero for small and large values. When computing the derivative of the error for a weight at the beginning of the network, you are multiplying a lot of sigmoid derivatives with one another (one for each layer). This leads to the so called “gradient-vanishing”, which leads to weights practically not being updated. Using the Relu (whose gradient is either zero or one) solves this problem entirely. You might see this if you try to increase the number of layers in the network that we created.

Q4: How do I set up the number of layers and neurons in each layer?
A4: Trial and error. It sounds crude and it is. Unfortunately, there is no simple way around it. We don’t know of systematic ways to set up the size of a neural network to solve a problem. The general rule of thumb is “the more complex the problem, the larger the network”, is not particularly helpful if you are just starting and you don’t have enough experience with NNs. In general, a simple and practical advice if you are trying to solve a specific problem is to look for something similar already solved by someone else and to try and use their network as an inspiration to dimension yours.

Q5: Why can we achieve the same accuracy by having only one layer? What is the advantage of having multiple ones?
A5: Again, it depends on the data. It is theoretically possible to represent every possible function using a single hidden layer, but for more complex datasets this might prove extremely difficult (in a similar fashion to convolutional networks, the more layer the higher the level of abstraction). The MNIST database is a very simple one and it is possible to do so, but in this specific example the focus was more on the fact that by using Pytorch we are able to create very complex networks with little effort.