MS2 Report

AIGUERPERSE Ambroise (341890),
GIRAUD Louis (334749), ROCHE Louis (345620)

1 Introduction

In this Milestone, we will implement and test a Multi-Layer Perceptron (MLP) and a Convolutionnal Neural Network (CNN). We will also implement the Principal Components Analysis (PCA). In the first part we will test the two classifiers and compare their best accuracy after tuning their parameters. In a second part we will test the impact of reducing the dimensionality of the data using PCA, before passing it to a classifier. We will try this on the MLP and SVM (one of our old classifier). The goal will be to compare the runtime and accuracy with or without the PCA preprocessing. We will evaluate these classifiers on the HASYv2 dataset, which consists of labeled black and white images of mathematical symbols, partitioned into 20 different classes. The data was normalized with the formula $x_{new} = \frac{x-\mu}{\sigma^2}$. The validation set used was made with a 80/20 ratio of the data set.

2 Neural Networks

2.1 MLP

For the MLP, we kept three fully connected layers, and tried to find an architecture, learning rate and max iteration to obtain a $> 90\%$ accuracy on the validation set (using a 80-20 split on the training set). We noticed that the number of nodes that worked the best was having each layer's size being half the previous one. The one that would give us the $> 90\%$ accuracy was $[256, 128, 64]$. We noticed that printing the loss along every iteration helped finding the right `max_iter` and `lr` parameters. When we saw that the loss was progressing too slowly, we increased the `lr` andif the execution stopped before the loss was low enough, we increased `max_iter`. This allowed us to get an accuracy on the validation set of 90.017%, with `lr= 1e-1` and `max_iter = 30`.

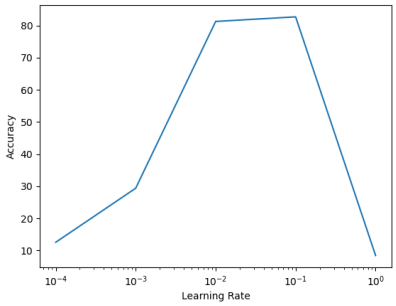

Figure 1 : Accuracy with different learning rates for MLP

Testing these parameters on the testing set yielded an accuracy of 85.121%, which is a bit lower but still not too bad. We still need to note that the filters are initialized randomly, meaning that the accuracy might differ for about $\pm 5\%$ from one run to the other.

2.2 CNN

For CNN, we sought the best hyper-parameters to obtain an optimal accuracy. First, we tried to find the best learning rate. Then, we plugged in the learning rate we found and sought for the optimal number of iterations. The default setup is the following: `lr = 1e-2, max-iter = 100`

We used the convolutionnal neural-network below.

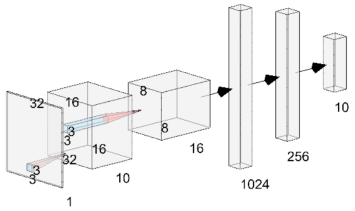

Figure 2 : Architecture of the CNN

It has two convolutionnal layers followed by a small MLP. Each convolution is max-pooled with a filter of size 2x2, which is why the image sizes are divided by 2 each time. To find the best learning rate, we plotted the accuracy on the validation set in function of the learning rate. We obtained the following plot:

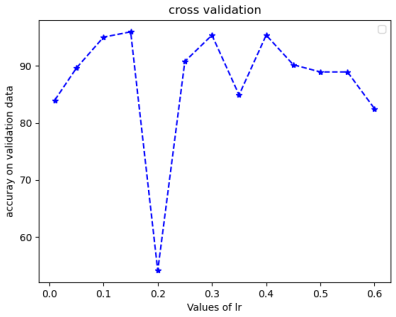

Figure 3 : Accuracies for different learning rates - CNN

On one hand, lower learning rates are insufficient for convergence due to low loss reduction, or getting stuck in local minima and being unable to escape from it. On the other hand, the higher learning rates also seem to struggle to find a minimum of the loss function, since the network may overshoot the optimal point and oscillate around it, or even diverge. Notably, we observe a downward spike at `lr=0.2`, suggesting that the optimizer overshot the loss. This indicates that learning rates of such magnitude and beyond are inconsistent and may result in divergence. The best accuracies seem to be 0.15, 0.2, 0.35, or 0.45, with respective accuracies of 95.04%, 95.96%, 95.39% and 95.39%. The plots exhibits that the accuracies after 0.2 learning rate are inconsistent as

they wobble up and down. In addition, 0.15 learning rate seems risky, as it is right next to a downward spike, which indicates that it might be too high. Nevertheless, the learning rates from 0.01 to 0.15 create a positive trend. So in-between 0.10 and 0.15, i.e. 0.125 seems to be the best learning rate. On the test set, it yields at best 93.86% accuracy, which is satisfactory and what was asked.

Then, we plugged in the best learning rate (0.125) and tried to find the most efficient number of iterations by plotting the values of the loss functions each iteration. We obtained the following plot:

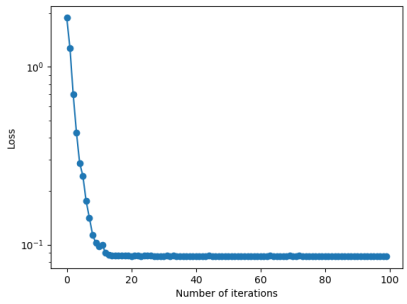

Figure 4 : evolution of loss with iterations - CNN

We see that there are significant gains during the first 10 iterations. Then, the loss starts to stagnate after 30 or so iterations. Thus, for our architecture of CNN, 30 iterations is the better parameter.

This couple of parameters yield $> 90\%$ accuracy, which satisfies the requirements. However, it differs significantly from the models in MS1 which required lower learning rates and higher maximum number of iterations. It is important to note that, for each time the data is fit by the CNN, the filters are initialized randomly. Thus, some seed will yield better results than others, meaning that for each settings, results may vary.

3 PCA

We tested PCA to reduce the dimension input to our classifiers. To choose the the parameter `pca_d`, we used the SVM classifier and plotted the accuracy with `pca_d` ranging from 20 to 200. We used SVM (and not MLP) because the produced accuracy has no random component so it helped up having a consistent graph.

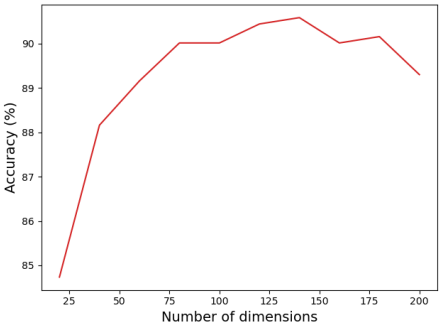

Figure 5 : Accuracy on SVM using different number of dimension with PCA

We observe that the best accuracy is obtained around 140. We therefore tested PCA on MLP and SVM with parameters `pca_d = 140`. This parameter an explained variance of 77%.
For the SVM, the execution time was reduced from 9.26s to 7s with the accuracy even going up from 90.3 to 90.6%. On the MLP, the number of trainable parameters dropped from 660k to 181k. The execution time reduced from 23s to 16s and the accuracy stayed about the same. We could probably make the execution faster by reducing `pca_d`, also reducing the accuracy. We kept `pca_d=140`, as we found it was a good trade-off between execution speed and accuracy. The accuracy was able to stay the same because the inputs are not random, but the symbols very likely live on some subspace of the original input dimension.

4 Bonus

As a bonus, we implemented a GUI where the user can draw a symbol and use our network to predict its label. To use it, call the main with the `--draw` argument (use only with no PCA and with MLP). For the application to work, the user might need to modify the rectangle's dimensions used to grab the image in the method classify handwriting. Using this GUI can gives insight on how the network classifies new images. For example, if we draw a mix of a triple bar and the approximately equal symbol by doing 3 wavy lines, the predicted label is the one of approximately equal, which seems to indicate that the network takes more consideration in the curvature of lines than in their numbers. Click here for a link to a presentation video of the bonus in case it doesn't work on your machine.

5 Conclusion

Our best model is the convolutionnal neural network. It yields more than 90% accuracy on the test set, whereas MLP does at best around 88%. It is not surprising, as the inputs are images and CNN, contrary to MLP, takes advantage of neighborhood relationships between pixels. It also has the advantage of having less parameters and weights, since the filters are translation equivariant, and thus less chance to overfit. This makes CNN more adapted for this project.