TP 6: Deep Learning

NPM3D - Mars 2022
Mehdi Zemni & Chady Raach
mehdi.zemni@student-cs.fr chady.raach@student-cs.fr

A. Understand MLP for geometric 2D data

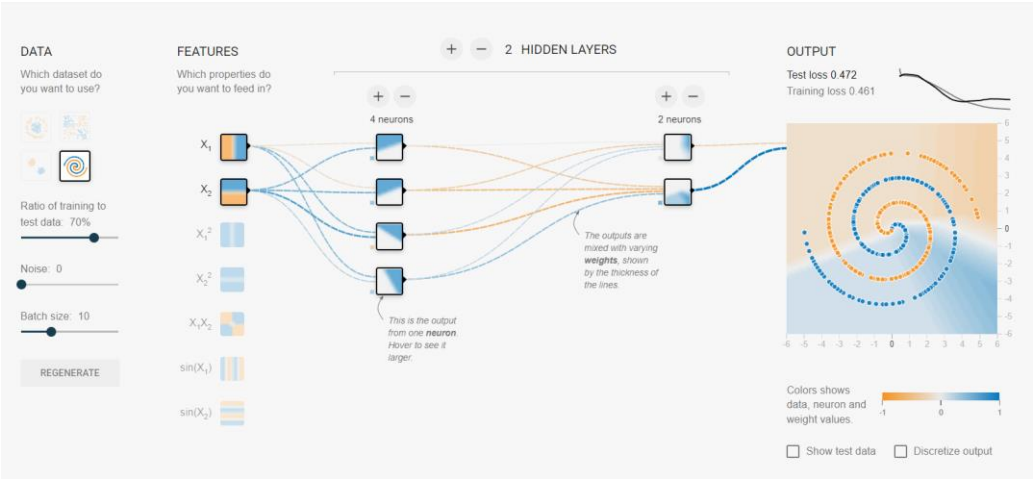

Figure 1: Experiment using playground tensorflow on 2D points (simple architecture)

We use playground tensorflow to do a classification on a 2D dataset of points forming a spiral. The network has a single hidden layer with a Relu activation, and it was trained with 10^{-3} learning rate. We can see the decision boundary formed on the right after training (fig;1). Unfortunately, this model is not complex enough to separate all points (classify correctly blue and orange points). Therefore, we decide to use a more complex architecture: we add new input variables, we increased the number of neurons and the network depth. In fig.2 we get a more complex decision boundary that separates all points.

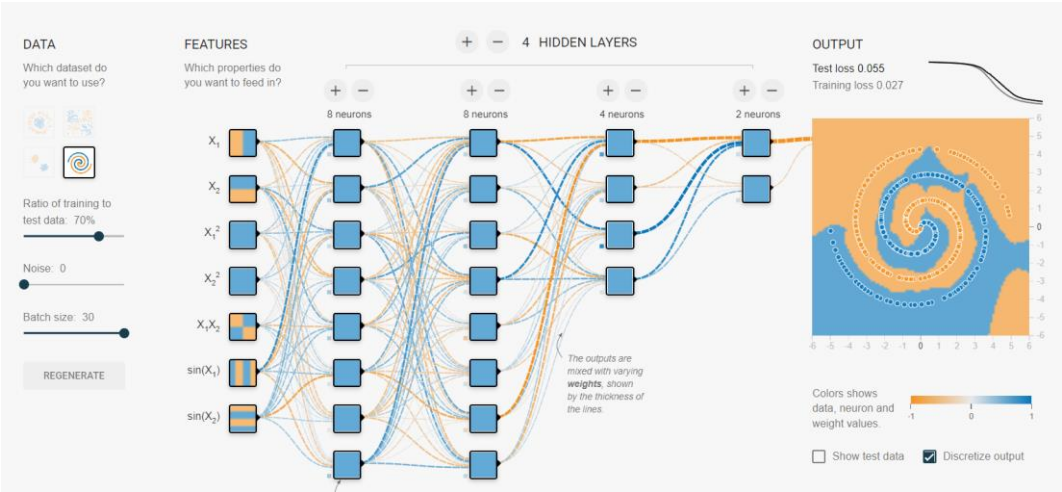

Figure 2: Experiment using playground tensorflow on 2D points (complex architecture)

B. PointMLP in PyTorch

All experiments are conducted on a limited number of epochs between 25 and 35 epochs due to limited computational capacity.

First, we implement the PointMLP architecture. It is a classical MLP architecture with linear layers and Relu activations.
This architecture presents 2 disadvantages: first it is not invariant to point shuffling, second, it operates on flattened arrays (all 3D coordinates are stacked in the same axis) => It doesn't capture spatial information like in the next architectures. In fig.3, we plot the evolution of the average loss (averaged over the epoch) and the accuracy (on test set) along the training (0.19 after 25 epochs).

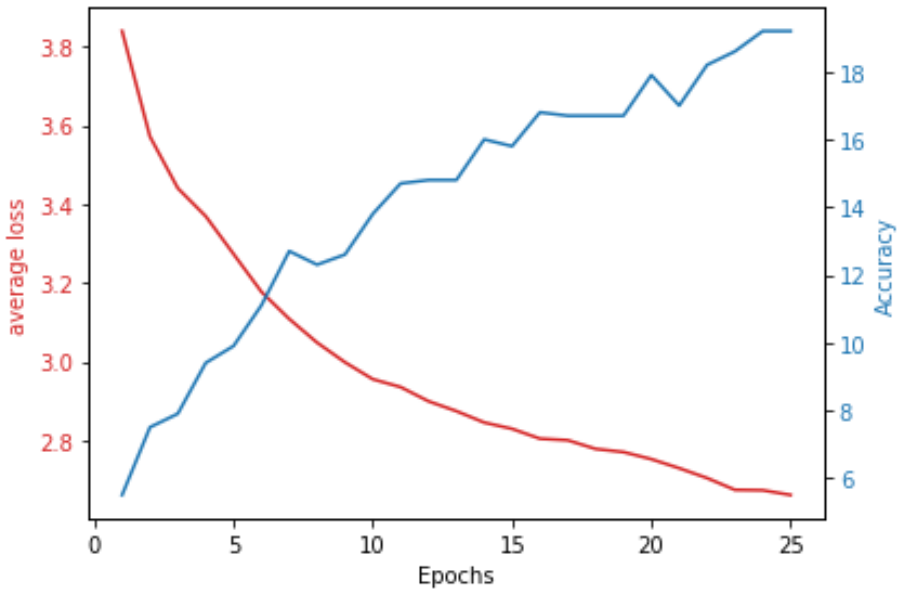

Figure 3 Accuracy and average loss for PointMLP on ModelNet40

C. PointNet in PyTorch

First, we implement the PointNetBasic without the T-Net modules. With this architecture we tackle the two problems we discussed in the previous question. Its is invariant to shuffling because all point coordinates will be multiplied with the same weights (we use a conv1d with kernel of size 1 that will do the same operation over the 1024 points). The spatial information is also kept because points are represented in 3D coordinates (3 channels) and these channels will be augmented through the network by adding more features. Accuracy and average loss are shown in fig.4. (We obtain 0.80 accuracy after 30 epochs (very large gab compared to PointMLP))

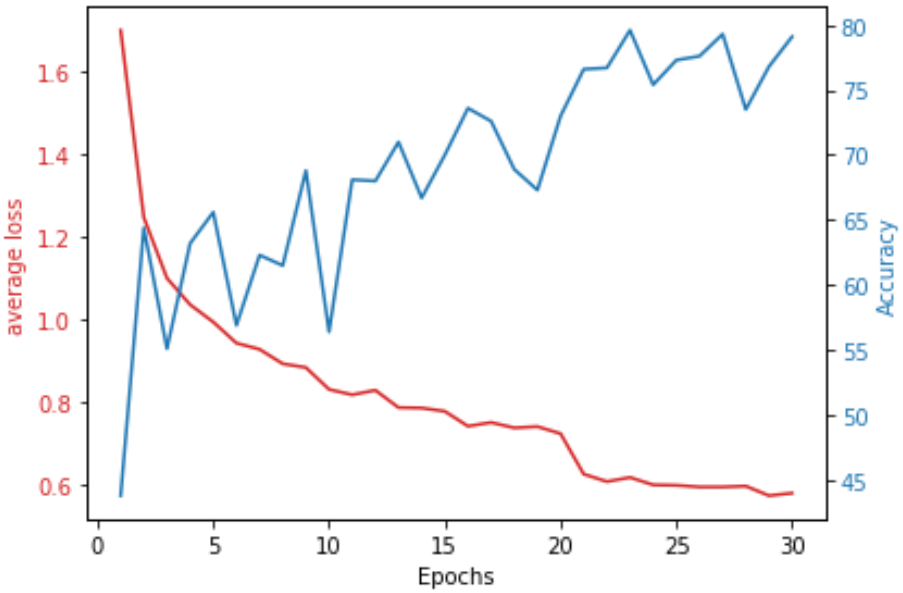

Figure 4 Accuracy and average loss for PointNetBasic on ModelNet40

In the second step we implement a full PointNet architecture by adding a 3x3 T-Net module. The goal of the T-Net module is to learn a transformation (a 3x3 matrix) which will be used to align the point cloud by simply multiplying it by this matrix. We also added a regularization loss to the softmax classification loss to make the matrix computed by the T-Net close to orthogonal. From our experiences, we found that this architecture achieved almost the same accuracy as the PointNetBasic but required longer training. Accuracy and average loss are shown in fig.5. (We obtain 0.74 accuracy after 35 epochs and 0.8 after 45 epochs)

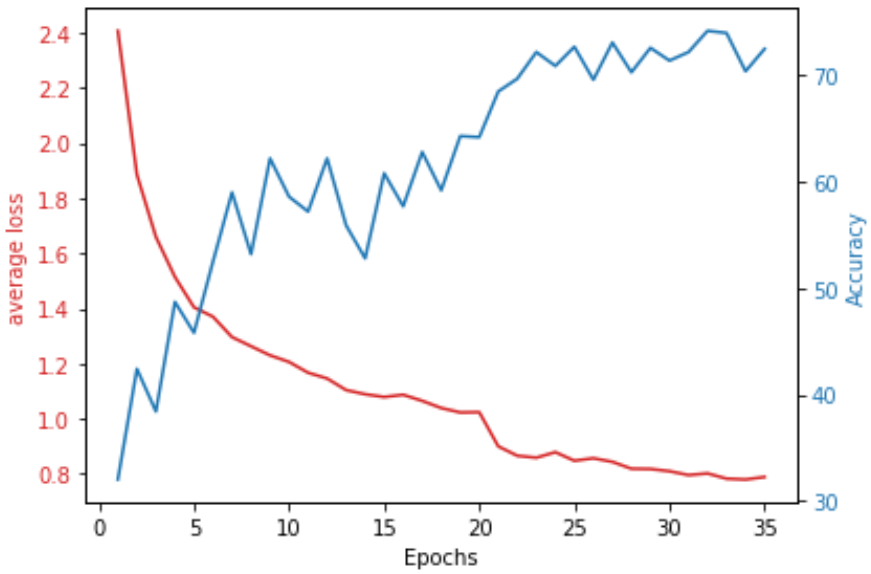

Figure 5 Accuracy and average loss for PointNetFull on ModelNet40

For the data augmentation, we added two transformations: a random symmetry with respect to one of the 3 axis and random anisotropic scale where we multiply the point cloud by a diagonal matrix containing 3 different diagonal values λ_1 λ_2 and λ_3 sampled randomly between 0.85 and 1.15. We trained PointNetBasic using these new augmentations on top of the default ones used before. The training was a bit slower in convergence (-0.05 accuracy at epoch 30), but accuracy results were close to previous results after convergence. We think that longer trainings (hundreds of epochs) can be useful to see more clearly the advantage of these augmentation (on preventing overfitting for example).

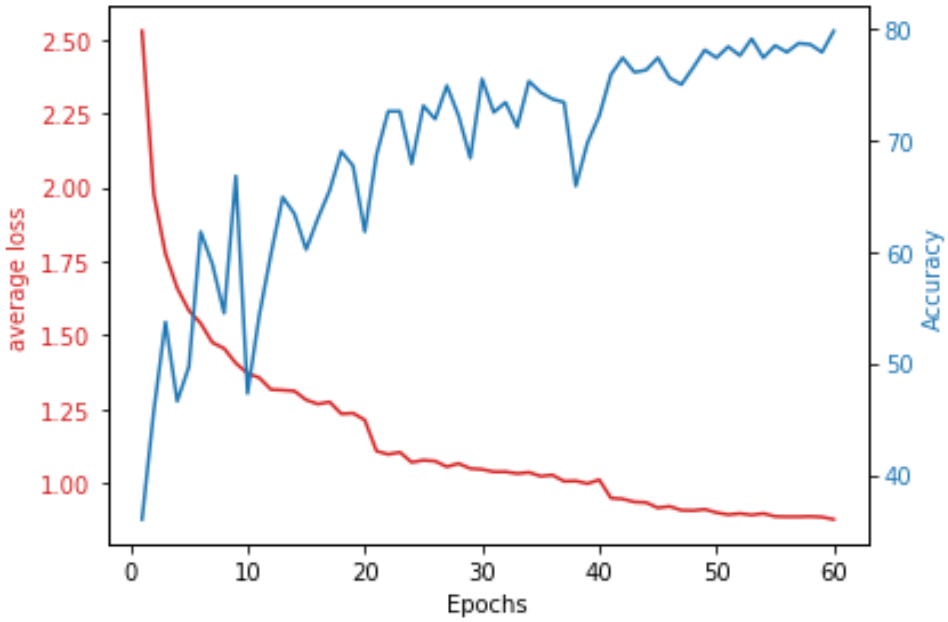

Figure 6 Accuracy and average loss for PointNetFull on ModelNet40 + augmentations