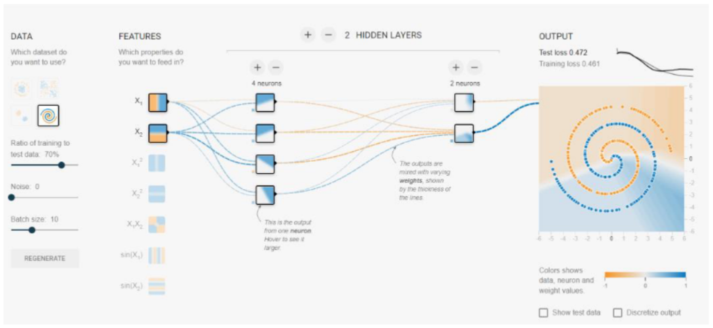
NPM3D - TP 6: Deep Learning

Adrien Golebiewski

IASD 2022-23

1 Understand MLP for geometric 2D data

To understand the behaviors of classical neural networks over 2D geometric data, I will first play with a neural network learnable on Internet across the web link : https://playground.tensorflow.org/


DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
X₁
X₂
X₁²
X₂²
X₁X₂
sin(X₁)
sin(X₂)
+ − 2 HIDDEN LAYERS
4 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Colors shows data, neuron and weight values.
-1 0 1
REGENERATE

Ratio of training to test data: 70%

Noise: 0

Batch size: 10

Test loss 0.472

Training loss 0.461

☐ Show test data ☐ Discretize output

Figure 1: Two planes extracted consecutively by RANSAC

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.

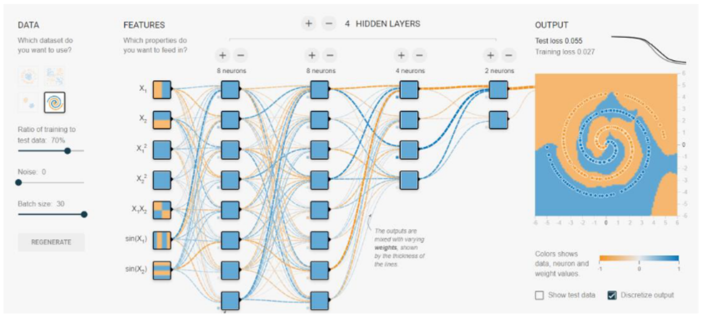

DATA
Which dataset do you want to use?
Ratio of training to test data: 70%
Noise: 0
Batch size: 30
REGENERATE
FEATURES
Which properties do you want to feed in?
4 HIDDEN LAYERS
8 neurons
8 neurons
4 neurons
2 neurons
OUTPUT
Test loss 0.055
Training loss 0.027
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.
Show test data
Discretize output

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

2 MLP on point clouds in PyTorch

Before implementing PointNet, we will first try a classical 3 layers MLP as neural networks to do the point cloud classification on ModelNet.

3 Question 1

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.
The table below sum up our test accuracy on ModelNet10 PLY or ModelNet40 PLY with our MLP neural network.

PointMLP	ModelNet10 PLY	ModelNet40 PLY
25 epochs	22.0%	15.20%
250 epochs	17.20%	25.10%

We can see that using the PointMLP model isn't efficient at all, even when we use a lot of epochs to train it. Indeed, this model isn't well fitted to this use : we only have fully connected layers, which means that the pointcloud is processed as a data field, and no geometrical features are taken into account. For example, two symmetric pointclouds wouldn't be processed the same way, which is done in the data preprocessing (when using rotations).

Thus, the accuracy of the network cannot be very high.

4 PointNet in PyTorch - Question 2

Secondly, we implemented the basic version of PointNet without the T-Net modules on ModelNet10 PLY or ModelNet40 PLY.
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.

The table below sum up our test accuracy on ModelNet10 PLY or ModelNet40 PLY :

PointNet Basic	ModelNet10 PLY	ModelNet40 PLY
25 epochs	88.5%	81.60%
250 epochs	90.2%	86.60%

On the ModelNet40 dataset, we achieve approximately 87 percent of accuracy, which is way better than the previous model. This is expected, because the maxpooling layer will re-orient the input vectors in an orientation-independant way.

The results with this "mini" PointNet version is already a significant improvement.

5 PointNet with T-Net network - Question 3

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.

The table below sum up our test accuracy on ModelNet10 PLY or ModelNet40 PLY :

PointNetFull	ModelNet10 PLY	ModelNet40 PLY
25 epochs	89.8%	84.4%
250 epochs	90.6%	86.70%

We can see that the results are a little bit better than with the previous model, but the gap isn't as high as expected. This can be due to the fact that the data augmentation isn't very significant : the T-Net network allows the network to be more resilient to affine transformations for example. This method required longer training.

We should then try to see if such data augmentations can have an impact on the performance (see Question 4).

6 Data augmentation for 3D data - Question 4

Finally, we augment the data by using random scaling and random translations as well to see if we can improve the results.

The table below sum up our test accuracy on ModelNet10 PLY or ModelNet40 PLY :

PointNetFull with augmentation	ModelNet10 PLY	ModelNet40 PLY
25 epochs	84.7%	81.5%

We can see that the results are worse than the previous one, nevertheless, there are still over 80 percent accurate, and we lost only about 5 percent of accuracy when the data was changed by a scale and translation factor.

However, when comparing to the PointNet Basic network (without the T-Net), we can see that the results got significantly worse :

PointNetFull with augmentation	ModelNet10 PLY	ModelNet40 PLY
25 epochs	84.7%	81.5%

To conclude, the PointNet Full network model is more resilient to the preprocessing, compared to the Basic version. The T-Net is thus fulfilling its job.

In my opinion, longer trainings (hundreds of epochs approximatively) can be useful to see more clearly the advantage of these augmentation (on preventing overfitting for example).