TP 6: Deep Learning

Objectives

- Understand MLP neural networks for geometric data
- Implement PoinNet network in Pytorch for 3D Point cloud classification

The report should be a pdf containing the answers to the **Questions** and named "TPX_LASTNAME.pdf". Your code should be in a zip file named "TPX_LASTNAME.zip". You can do the report as a pair, just state both your names inside the report and in the pdf and zip filenames, like "TPX_LASTNAME1_LASTNAME2.pdf"
Send your code along with the report to the email mva.npm3d@gmail.com. The object of the mail must be "[NPM3D] TPX LASTNAME" or "[NPM3D] TPX LASTNAME1 LASTNAME2" if you are a pair working on the report.

The goal of that practical session is to implement and test a specific and well-known architecture of neural network for 3D Point Clouds: PointNet.

We will apply PointNet on a classical task for 3D data: point cloud classification. A well-known dataset for that task is Princeton ModelNet: https://modelnet.cs.princeton.edu/. The goal is to classify point clouds in categories like chair, sofa, airplanes...

Normally, the Princeton ModelNet is made of CAD models as meshes. You will find in the data repository ModelNet10_PLY and ModelNet40_PLY (see Figure 1) point clouds with 1024 points sampled over the meshes of ModelNet10 (10 classes) and ModelNet40 (40 classes). We recommend you to work first on ModelNet10_PLY for debugging, especially if you have no GPU. ModelNet40_PLY is here for you to compare your results to the original results of the PointNet paper: https://arxiv.org/pdf/1612.00593.pdf

In Deep Learning, you need GPU for training neural networks efficiently. If you do not have access to GPU, you can use Google Colab (online notebook with free access to GPU for 12 hours).

Figure 1 A point cloud of an airplane in the dataset ModelNet40_PLY

A. Understand MLP for geometric 2D data

To understand the behaviors of classical neural networks over 2D geometric data, you will first play with a neural network learnable on Internet (Figure 1).
Go the webpage: https://playground.tensorflow.org/

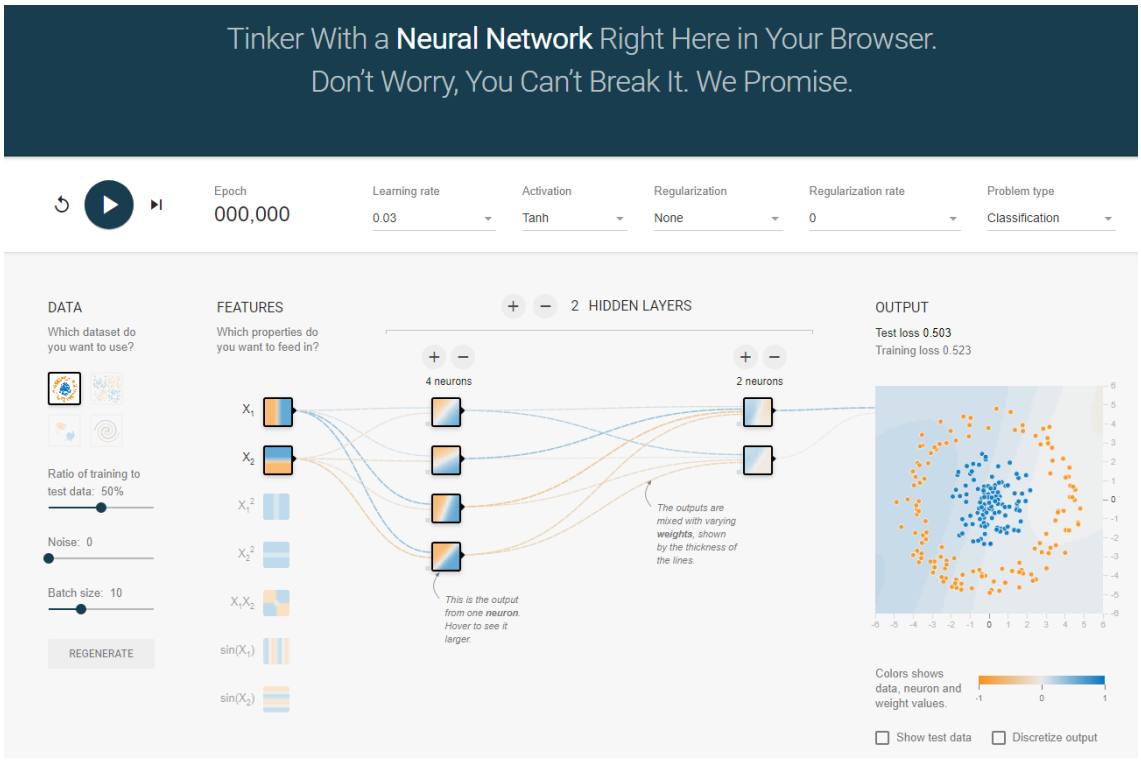

Figure 2 Website to play with a neural network on 2D data

You can change number of parameters before training: input data, noise, learning rate, activation functions, number of hidden layers.

We recommend using:

- 0.001 as learning rate
- ReLU as activation functions

What is happening for the 2D spiral input point cloud?

B. 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. The architecture is:

- First, the point cloud is flatten to get 1024 points * 3 (x,y,z) = 3072 inputs.
- The first layer is Linear(3072,512)
- The second layer is Linear(512,256) with a weight dropout of p = 0.3
- The last layer is Linear(256,N) with N the number of classes

The first and second layers have batch normalization and ReLU as activation functions.

Starting from the file pointnet.py, complete the class MLP(nn.Module) to implement the neural network as defined above. The Data Loader and data augmentation are already implemented. The cross entropy classification loss is already in the basic_loss() function. The training procedure is also implemented in the function train().

You will need nn.Linear() for linear weights, nn.BatchNorm1d() for batch normalization and nn.Dropout() for weight dropout. You will use nn.Flatten() to flatten tensors, and F.relu() for the ReLU function activation.

In pointnet.py the default number of epochs is 250 (as in the original experiments of PointNet). If it takes too much time on your computer, you can go down to 10 epochs. In that case, keep the same number of epochs in all experiments of the practical session (and give the number of epochs in your report).

Question 1: Give your test accuracy on ModelNet10_PLY or ModelNet40_PLY with your MLP neural network. Give your parameters (learning rate, nb epochs). Comment the results.

C. PointNet in PyTorch

1. Basic version of PointNet

PointNet is a simple but very efficient network able to learn features over point clouds.

You can see in Figure 2 the PointNet architecture from the original paper.

The full architecture is decomposed in 6 steps:

- First a T-Net network output a 3*3 matrix to align 3d Point clouds
- A 2 layers shared MLP over points to compute features of dim 64:
  - Linear(3,64)
  - Linear(64,64)
- A new T-Net network output 64*64 matrix to align points in feature space of dim 64
- A 3 layers shared MLP over points to compute features of dim 1024:
  - Linear(64,64)
  - Linear(64,128)
  - Linear(128,1024)
- A symmetric function (max pooling) to compute a global feature of dim 1024
- Finally, 3 layers MLP from the global feature to output scores for each class
  - Linear(1024,512)
  - Linear(512,256) with a weight dropout of p = 0.3
  - Linear(256,N) with N the number of classes

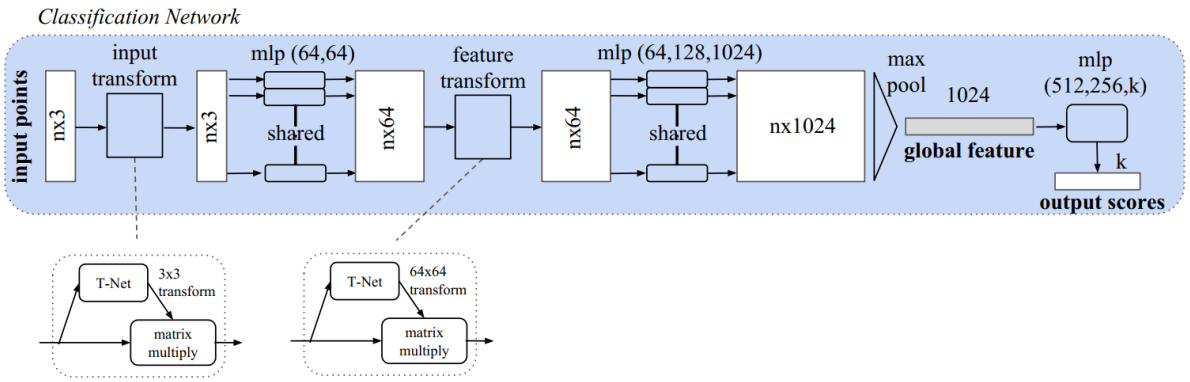

Figure 3 PointNet architecture (from the original paper)

You will first implement a basic version of PointNet without the T-Nets.

In the file pointnet.py, complete the class PointNetBasic(nn.Module) to implement the basic version of PointNet without T-Net networks. All layers, except the last one include Linear, then batch normalization, and then ReLU. You will use the basic_loss() function.

You will need nn.Conv1d() functions to do shared Linear over points, nn.Linear() for the global MLP, nn.BatchNorm1d() for batch normalization and nn.Dropout() for weight dropout. You will use nn.MaxPool1d() to compute max pooling, nn.Flatten() to flatten tensors, and F.relu() for the ReLU function activation.

Question 2: Give your test accuracy with the basic version of PointNet on ModelNet10_PLY or ModelNet40_PLY. Give your parameters (learning rate, nb epochs). Comment the results.

2. PointNet with T-Net network

To improve accuracy of PointNet, it is possible to align 3D point clouds before feeding the points to the network. You will implement a version of PointNet with only the first T-Net that outputs a 3*3 matrix.

T-Net is in fact a mini-PointNet that takes raw point cloud as input and regress a 3*3 matrix. T-Net is decomposed into 3 steps:

- A 3 layers shared MLP:
  - Linear(3,64)
  - Linear(64,128)
  - Linear(128,1024)
- A max pooling across points to compute a global feature of dim 1024
- A 3 layers global MLP:
  - Linear(1024,512)
  - Linear(512,256)
  - Linear(256,k*k) with k the dimension of the matrix

In the file pointnet.py, complete the class Tnet(nn.Module) to output a k*k matrix. The output matrix is added to an identity matrix to simplify the learning. All layers, except the last one include Linear, then batch normalization, and then ReLU.

In the file pointnet.py, complete the class PointNetFull(nn.Module) to implement a version of PointNet with the first T-Net network. You will use the pointnet_full_loss() with a regularization loss added to the cross entropy classification loss (make the matrix computed by the T-Net close to orthogonal).

Question 3: Give your test accuracy with PointNetFull adding the 3*3 T-Net on ModelNet10_PLY or ModelNet40_PLY. Give your parameters (learning rate, nb epochs). Comment the results and compare to the basic version results.

3. Data augmentation for 3D data

Data representation and data augmentation are 2 keys components for 3D deep learning, as much as architectures and corresponding losses.

In pointnet.py, we used two classical data augmentation with random rotation around z, and random Gaussian noise on points.

Question 4: Find a new data augmentation on 3D point clouds. Explain your idea. Give your test accuracy with and without your data augmentation on ModelNet10_PLY or ModelNet40_PLY (using basic or full version of PointNet). Give your parameters (learning rate, nb epochs). Comment your results.

A. Going further (BONUS)

PointNet is efficient but do not use neighborhood information and is limited to small point clouds.

Other well-known neural networks for 3d data are based on sparse tensors. They are able to do efficient voxel convolutions of 3d data to aggregate information of neighborhoods and can be applied on much larger point clouds. MinkowskiEngine is an efficient implementation of sparse tensors in PyTorch.

Download the github https://github.com/NVIDIA/MinkowskiEngine and try to do a classification on ModelNet.

Question Bonus: **Try MinkowskiEngine on ModelNet10_PLY or ModelNet40_PLY with the same data augmentation that for the experiments with PointNet. Use the MinkowskiFCNN model (available in examples/classification_modelnet40.py). Explain your protocol. Give the test accuracy you get. Give your parameters (learning rate, nb epochs). Comment the results and compare to PointNet results.**