Master PAR – 3D Deep Learning

Lab – Neural Network for 3D Point Clouds: PointNet

MLP Neural Networks for Geometric Data – 3D Point Cloud Classification

Academic year 2025–2026 Jean-Emmanuel Deschaud – Paul Checchin

---

Ojectives

The objectives of this lab are:

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

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

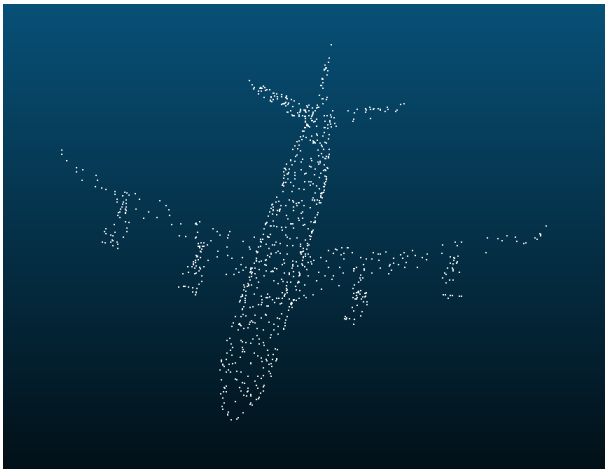

Figure 1: A point cloud of an airplane in the dataset `ModelNet40_PLY`.

PointNet provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing. It directly takes point clouds as input and outputs either class labels for the entire input or per point segment/part labels for each point of the input. 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...

Initially, the Princeton ModelNet dataset 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).

Deliverables

This lab contains a number of **questions** that require your response. In your lab writeup, you should restate the question and then provide your answer. In short, your writeup should be comprehensible without needing to reference this document.

Submitting your writeup

Your lab writeup must be emailed in the following format:

```
TO: paul.checchin@uca.fr
SUBJECT: [3DML] PointNetLab: <uniqname1>, <uniqname2>
CC: <all of your team members>
```

Your email should contain exactly two attachments:

1. A PDF of your writeup. Equations must be typeset. No other formats (including .doc, .docx) are acceptable. Latex is always appreciated, but not required. Note: Your PDF must be less than 5MB in size. This is generally easily achievable by using images of sensible resolution. If this is impossible, put your PDF on a web server and include in your email a URL to the file.

   The report should contain the answers to the **Questions** and be named "`3DML_LASTNAME.pdf`". Your code should be in a zip file named "`3DML_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 "`3DML_LASTNAME1_LASTNAME2.pdf`"

2. A photo of your collaboration certification, signed by all members. It may be hand-written, and be no more than 200kB in size.

Deviations from this format will be penalized. If you receive a "moderation pending" message, it is harmless.

TEAMWORK

As a reminder, our class collaboration policy requires your team to work together on each problem. You may not divide up the work and proceed in parallel.

Exercice 1 PointMLP in PyTorch

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

The architecture is:

(a) First, the point cloud is flatten to get 1024 points $\times$ 3 (x, y, z) $=$ 3072 inputs;

(b) The first layer is `MLP(3072, 512)`;

(c) The second layer is `MLP(512, 256)` with a weight dropout of $p = 0.3$;

(d) The last layer is `MLP(256, N)` with N the number of classes;

(e) Use `logsoftmax()` to compute the score for each class.

(f) Each layer has batch normalization and `ReLU` as activation functions.

Starting from the file `pointnet.py`, complete the class `PointMLP(nn.Module)` to implement the `PointMLP` network as defined above. The `Data Loader` and data augmentation are already implemented. The `softmax` 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, `F.relu()` for the `ReLU` function activation and `nn.LogSoftmax()` for class score computation.

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 25 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).

! 1. Give your test accuracy on `ModelNet10_PLY` or `ModelNet40_PLY` with your `PointMLP` neural network.

2. Comment the results.

Exercice 2 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.

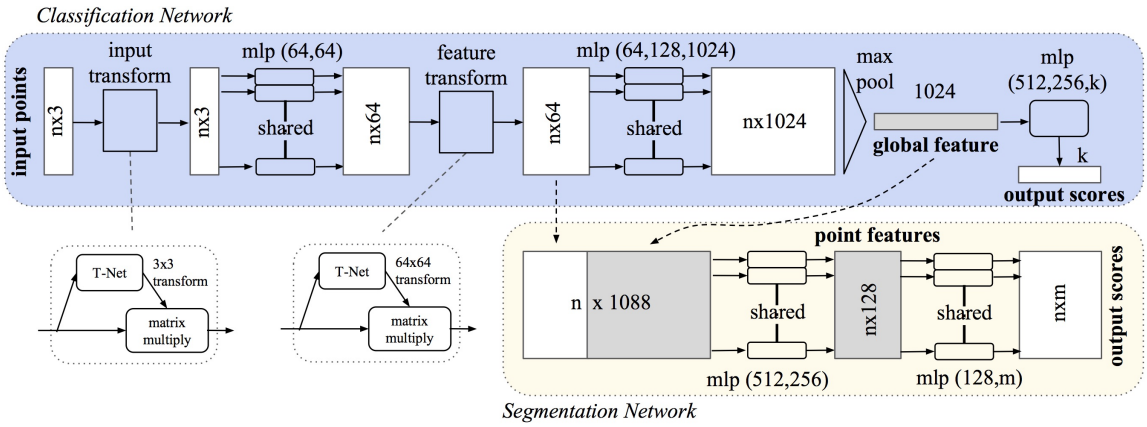

Figure 2: PointNet architecture (from the original paper). The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification score for m classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. MLP stands for Multi-Layer Perceptron, the numbers in bracket are its layer sizes. `Batchnorm` is used for all layers with `ReLU`. Dropout layers are used for the last MLP in classification net.

The full architecture is decomposed in 6 steps:

(a) First a T-Net network output a 3×3 matrix to align 3D point clouds;

(b) A 2 layers shared MLP over points to compute features of dim 64:
   i. `MLP(3, 64)`
   ii. `MLP(64, 64)`

(c) A new T-Net network output 64×64 matrix to align points in feature space of dim 64;

(d) A 3 layers shared `MLP` over points to compute features of dim 1024:
   i. `MLP(64, 64)`
   ii. `MLP(64, 128)`
   iii. `MLP(128, 1024)`

(e) A symmetric function (max pooling) to compute a global feature of dim 1024

(f) Finally, 3 layers MLP from the global feature to output scores for each class:
   i. `MLP(1024, 512)`
   ii. `MLP(512, 256)` with a weight dropout of $p = 0.3$
   iii. `MLP(128, 1024)` with N the number of classes

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 `ReLU` and batch normalization. You will use the `basic_loss()` function.

You will need `nn.Conv1d()` functions to do shared MLP 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, `F.relu()` for the `ReLU` function activation and `nn.LogSoftmax()` for class score computation.

! 1. Give your test accuracy with the basic version of PointNet on `ModelNet10_PLY` or `ModelNet40_PLY`.
2. Comment the results.

3. 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) A 3 layers shared `MLP`:
   i. `MLP(3, 64)`
   ii. `MLP(64, 128)`
   iii. `MLP(128, 1024)`

(b) A max pooling across points to compute a global feature of dim 1024;

(c) A 3 layers global `MLP`:
   i. `MLP(1024, 512)`

ii. `MLP(512, 256)`

iii. `MLP(256, k×k)` with k the dimension of the matrix

In the file `pointnet.py`, complete the class `Tnet(nn.Module)` to output a $k \times k$ matrix. The output matrix is initialized as an identity matrix. All layers, except the last one include `ReLU` and batch normalization.

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 softmax classification loss (make the matrix computed by the T-Net close to orthogonal).

! 1. Give your test accuracy with PointNet adding the 3×3 T-Net on `ModelNet10_PLY` or `ModelNet40_PLY`.

2. 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 three classical data augmentation with Random Rotation around z, shuffle of points and Gaussian noise.

! 1. Find a new data augmentation on 3D point clouds. Explain your idea.

2. Give your test accuracy with and without your data augmentation on `ModelNet10_PLY` or `ModelNet40_PLY` (using basic or full version of PointNet as you wish).