Balthazar Neveu — Jamy Lafenetre
balthazarneveu@gmail.com —
jamy.lafenetre@ens-paris-saclay.fr

Lab session # 6
NPM 2024

03/06/24

Preliminary question: Ability of MLP to classify coordinates.

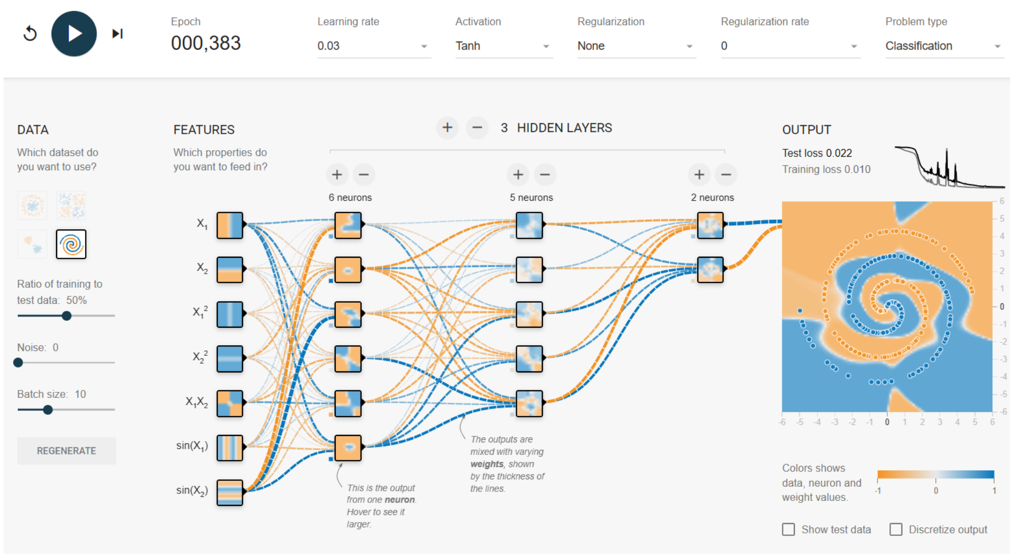
Epoch 000,383
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
3 HIDDEN LAYERS: 6 neurons, 5 neurons, 2 neurons
Test loss 0.022
Training loss 0.010
DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
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
Show test data
Discretize output

Figure 1: Multi layer perceptron show the ability to classify coordinates. We display parameters which led to good results on the spiral classification.

- The Cybenko theorem of universal approximation states that we should be able to classify this point cloud with a single hidden layer with enough neurons.
- When we use ReLu function, MLP outputs a piecewise linear function of the input coordinates.
- To classify the spiral points, the output logits may need to use the radius... which is possible to approximate with a very wide hidden layer.
- Hower instead of learning to appoximate $x^2 + y^2$ with a bunch of layers and ReLU to get the radius, we can directly inject it as an extra input.
- Introducing squared coordinates (and sinus embeddings) allows the model to output piecewise linear functions of more complex inputs features (a piecewise quadratic form if we consider the squared x^2).
- We note that finding the right architecture (depth and number of layers) is a matter of trial and error here.

We can see that eventually, we're able to find correct parameters which allow finding a shape which roughly match the spiral.
Now the following lab session deals on how we classify 3D point clouds (e.g. unordered sets of points in 3D space). We will use a PointNet architecture to classify 3D point clouds.

Question 1

We train our model on ModelNet40. It has 1.708M parameters. Using a learning rate of 10^-2, we are only able to reach an accuracy of 12% after 75 epochs (convergence is achieved).
The accuracy of a random classifier would be 2.5%, so this result is not dramatic. However it cannot be considered a good classifier. Please note that MLP considers that the order of the points matters (a list instead of a set) and therefore the output is not **permutation invariant**. A detailed unitary test is available in the code (test_pointnet.py to make sure that pointNet architecture is actually permutation invariant).

Question 2

On ModelNet40, we reach an accuracy of 86% after 75 epochs using a learning rate of $1e-3$. These results are considerably higher than using a simple MLP. A Pointnet without Tnets can be used like a satisfactory classifier. Unlike the initial MLP, PointNet processes each point of the set separately until the max pooling operator is reached. Max pooling statisfies the permutation invariance.

Question 3

On ModelNet40, we reach an accuracy of 84% after 100 epochs using a learning rate of $5e-3$. These results are slightly under what could be obtained without Tnet. Our intuition is that by using TNet, network may learn to orient objects in a canonical way, which may not always the best way to classify them... it is almost like trying to counter-act the effect of augmentations.

Question 4

We propose two sources of data augmentation.

- First, we apply three random scaling factors centered around 1 in the x, y and z direction.
- Second, we remove a random proportion of the points (around 10%) and replace them by duplicating other points.

Using the same parameters as before, the impact of this data augmentation is not meaningful and seems to slightly hinder the accuracy.

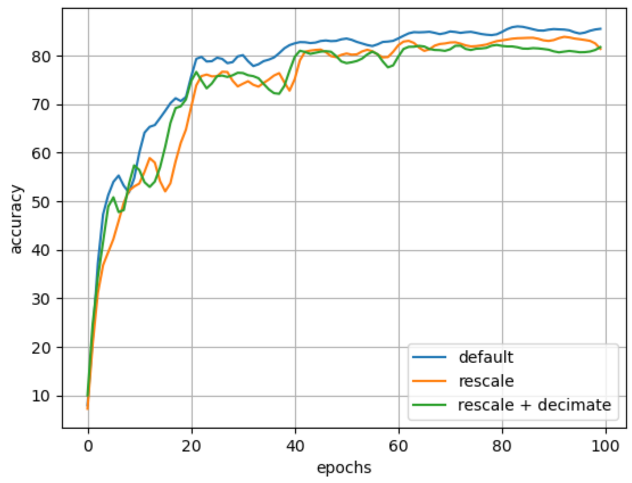

Figure 2: Impact on validation accuracy of our augmentations during PointNet training - seems like adding new augmentations sligtlty slowed down convergence and prevented to reach the best accuracy.

Conclusion

All evaluations conducted on ModelNet40 are summarized in Table 1. We can see that the PointNet architecture is able to classify 3D point clouds with a decent accuracy. However, the use of T-Net (promising idea on paper) does not seem to improve the results... nor data augmentation.

Model	Learning Rate	Epochs	Accuracy
MLP (Not permutation invariant)	10^{-2}	75	12%
PointNet without T-Net	10^{-3}	75	86%
PointNet with T-Net	5×10^{-3}	100	84%
PointNet with T-Net and Data Augmentation	5.10^{-3}	100	83%

Table 1: Summary of Point Cloud Classification Results

Foreseen potential ideas for improvements include:

- Add sinusoidal and quadratic embeddings of the coordinates into the input features.
- Interpretability of the PointNet architecture: visualize the points which were selected by the max pooling operator (find all max indexes and display the corresponding points in colors, we may find gain interpretability on how PointNet intuitively works).

Appendix: training curves

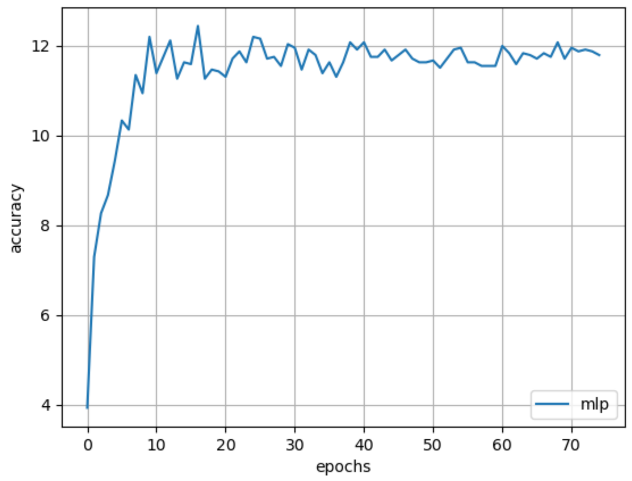

Figure 3: MLP accuracy during training

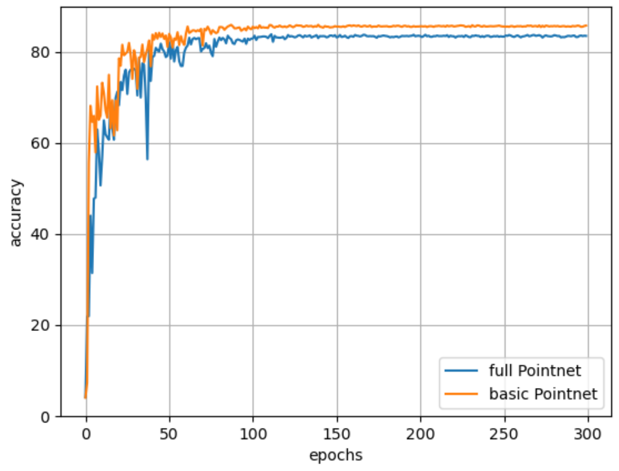

Figure 4: PointNet (no TNET vs TNET) + standard augmentations) validation accuracy during training

Appendix: T-NET validation

To validate our TNET implementation, we create a simple synthetic object 5. We then optimize (overfit) the TNET to ”correct” the orientation of the object. Convergence is demonstrated in figure 6. Please refer to the code check_tnet.py for more details.

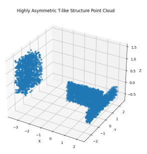

Figure 5: The expected orientation of the object

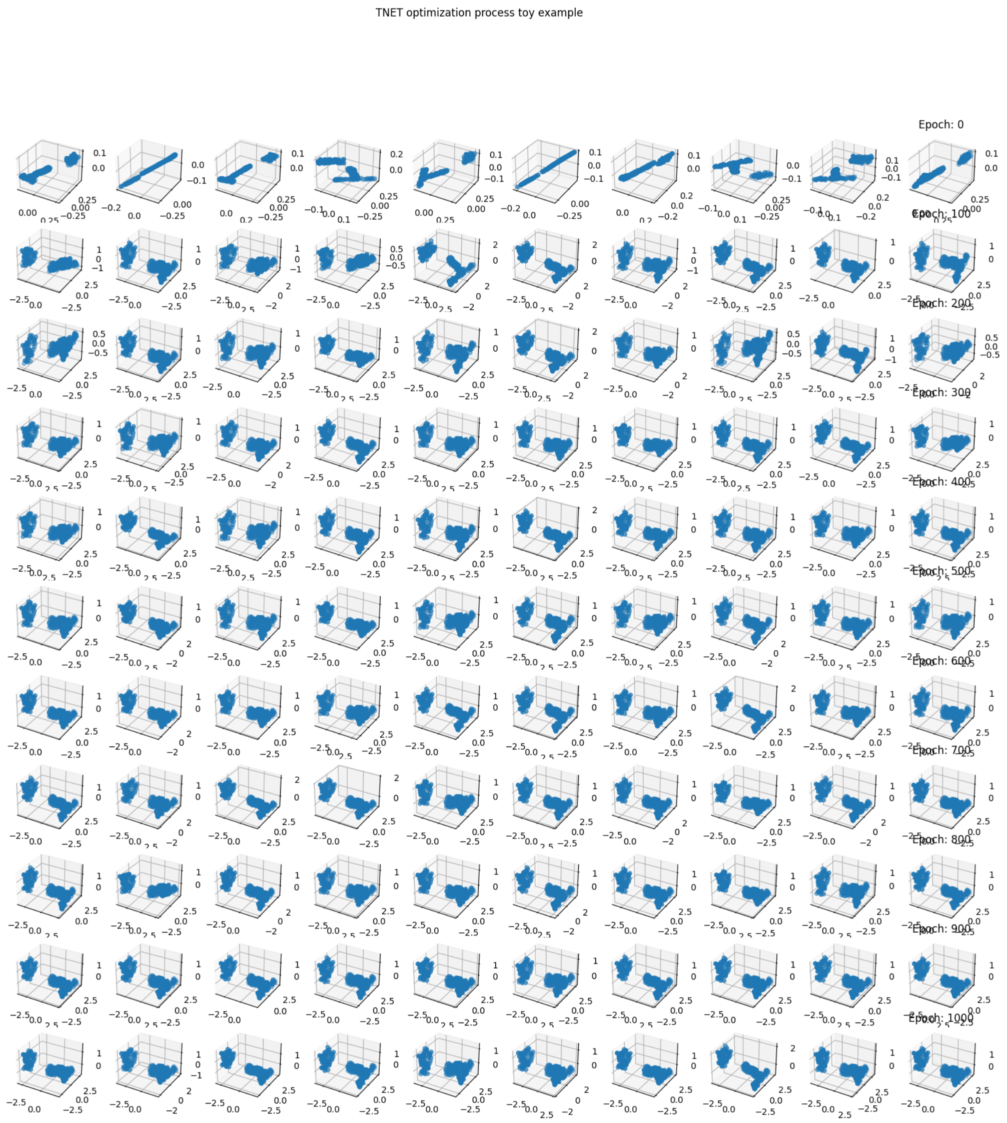

Figure 6: Horizontally, we show 10 same random orientations of the object after being ”corrected” by the TNET. Vertically, we see the convergence of the TNET across epochs. Here the optimization is performed on the predicted matrices directly and by optimizing the Froebenius norm of the difference between the predicted and expected matrices. We can see for instance that at the begining, the scales are not correct.

Please note that optimization could very well be performed on the predicted point cloud directly instead of the *rotation matrices.*