PMLDL'21. Project Progress D1.4

Student Information

Student: Anna Boronina

Group: BS18-DS-01

Telegram: @whoorma

The idea: using GAN for pointcloud generation

Progress

Recap: Originally, I was creating a GAN to generate point clouds. In the last progress report I showed that it didn't work, so I tried to create an autoencoder.

Data Augmentation

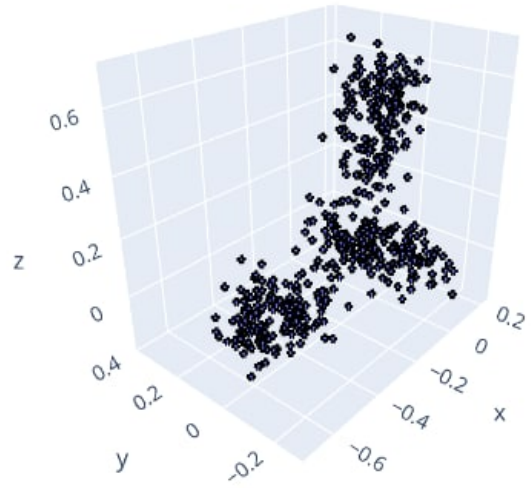
1. One of my mistakes was in data augmentation. I applied transformation during dataset creation instead of during training. So, even though dataloader shuffled all my data, the data was always the same. It lead to:
    1. Generator couldn't learn from it, so I fixed it and data augmentation (mainly, point sampling, sorting and normalization) is now done during the training, so it's very much different every time.
    2. Unlike GAN, autoencoder didn't like this approach because, as turned out, it overfitted on old training data and doesn't learn on newly augmented data at all.

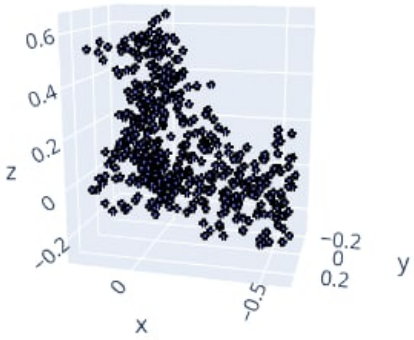
GAN

1. For both, Generator and Discriminator, I use Adam optimizer with learning rate of 0.0002. If I change it to 0.0001 everything stops working.
2. I simplified the Discriminator because it was learning much faster than the generator. It was very good at distinguishing between real and fake, hence didn't give Generator a chance to learn
3. Now generator consists of blocks like this one:

    ```
    nn.Linear(N, M) -> nn.LeakyReLU(0.2) -> nn.Dropout(0.2)
    ```

4. What the generator learned:
    1. Bed and Chair. It's a very simple architecture so I doubt it would ever learn more but this result.





Autoencoder

As I mentioned, there are two types of data: with augmentation done before training and during training. When it's trained on first type of data (augmented before training) it overfits and produces good results for samples from train set and bad results for samples from validation set.

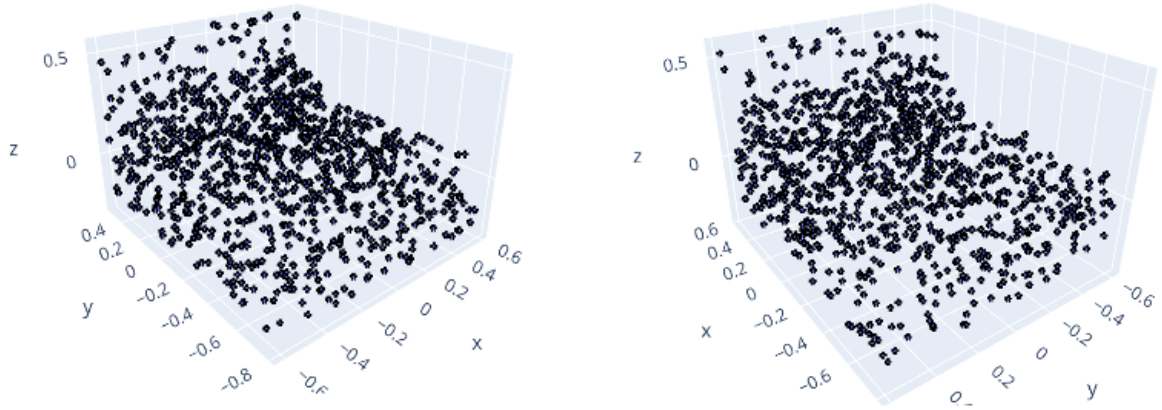
Right now encoder consists of `Conv1d` and `Linear` layers, decoder consists of `Linear` layers:

```
PointNetAE(
  (encoder): PointEncoder(
    (convs): Sequential(
      (0): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
      (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
      (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU()
      (6): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
      (7): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (dense): Sequential(
      (0): Linear(in_features=1024, out_features=512, bias=True)
      (1): ReLU()
      (2): Linear(in_features=512, out_features=50, bias=True)
    )
    (mu_fc): Linear(in_features=50, out_features=50, bias=True)
```

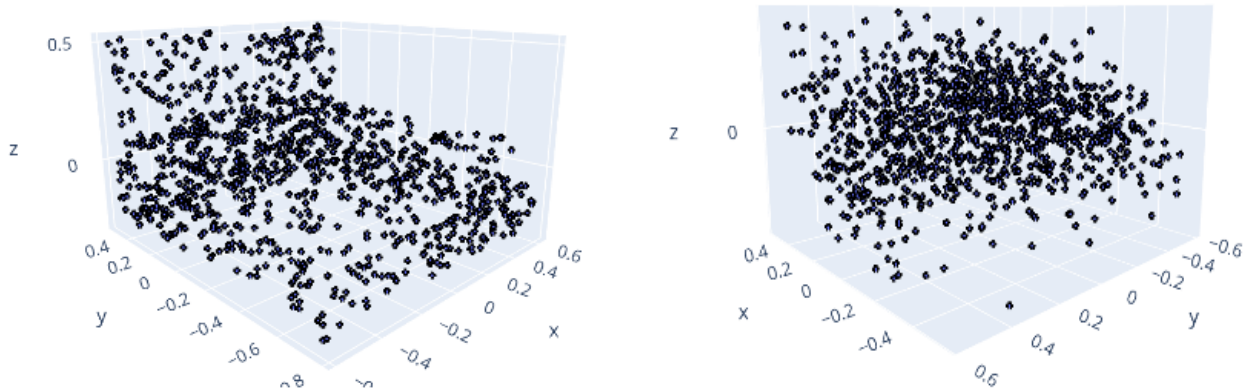
```
    (log_var_fc): Linear(in_features=50, out_features=50, bias=True)
  )
  (decoder): PointDecoder(
    (dense_layers): Sequential(
      (0): Linear(in_features=50, out_features=256, bias=True)
      (1): Dropout(p=0.1, inplace=False)
      (2): ReLU()
      (3): Linear(in_features=256, out_features=512, bias=True)
      (4): Dropout(p=0.2, inplace=False)
      (5): ReLU()
      (6): Linear(in_features=512, out_features=1024, bias=True)
      (7): Dropout(p=0.3, inplace=False)
      (8): ReLU()
      (9): Linear(in_features=1024, out_features=3072, bias=True)
      (10): Dropout(p=0.3, inplace=False)
      (11): Tanh()
    )
  )
)
```

Overfitted autoencoder

Original bed from train set and reconstructed



Original bed from validation set and reconstructed



Currently, the problem with the autoencoder is that it overfits on training data and produces something close to noise. To solve overfitting problem, I will try this:

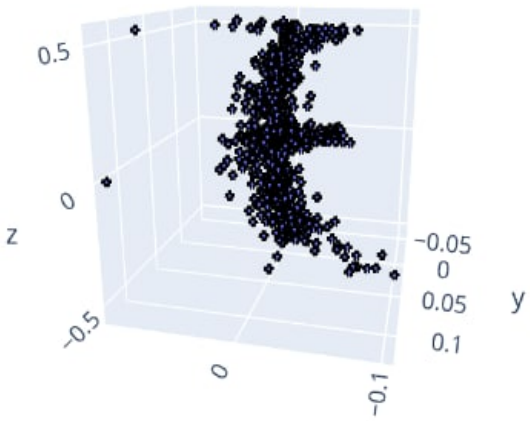
1. Keep the old augmentation (because it speeds up the training process a lot!) and introduce dropout-layers

2. If 1st doesn't work, I'll try to simplify the decoder a little bit.
3. If 2nd doesn't work, I'll try to add `TransposeConv1d` layers to the decoder.
4. Test on validation set during training to see when exactly it starts overfitting.

Not-overfitted Autoencoder

With data augmented during training, training process just gets stuck at some number and produces the following (it continues for several more epochs):

```
1:      train loss: 37.868529319763184
        validation loss: 36.3890380859375
2:      train loss: 36.71415328979492
        validation loss: 35.26276397705078
3:      train loss: 35.80773639678955
        validation loss: 35.20555877685547
4:      train loss: 34.94113874435425
        validation loss: 34.76276779174805
5:      train loss: 34.56571626663208
        validation loss: 34.612247467041016
6:      train loss: 34.40035581588745
        validation loss: 34.13066101074219
7:      train loss: 34.31328105926514
        validation loss: 33.59748458862305
8:      train loss: 33.939207553863525
        validation loss: 34.06801223754883
9:      train loss: 34.02021646499634
        validation loss: 33.80063247680664
10:     train loss: 33.88559103012085
        validation loss: 33.56843948364258
11:     train loss: 33.53111457824707
        validation loss: 33.22277069091797
```



I've tried changing optimizers and number of points for pointclouds (1024, 256, ...). Maybe it's an effect of too small or too large latent space size - it's to be figured out.

Links

Autoencoder training

Data preprocessing and dataloaders creation

Folder with everything project-related

What's next

1. Figure out why autoencoder is not training on data augmented during training
2. Use Xavier initialization as mentioned here
3. Try adding `Conv1d` layers to the decoder
4. Increase latent space (now it's 100 neurons)
5. Test if overfitting autoencoder can be turned into a VAE
6. If AE stops overfitting, turn it into a VAE