PMLDL'21. Project Progress D1.5

Student Information

Student: Anna Boronina

Group: BS18-DS-01

Telegram: @whoorma

The idea: using Variational Autoencoder for pointcloud generation

Progress

The sections below contain information about changes I introduced to my model and the training process. These changes are based on "What's next" section from the previous report.

Data Augmentation

I introduced new way to augment data, similar to how we augment images.

The problem before:

1. I had a set of pointclouds, each containing around 3000 points. I sampled 1024 from them during the training process, so my model has probably never seen the same pointcloud more than once. So, the training was very slow.
2. Another approach was to apply all transformations before training. It caused overfitting because the model saw the same pointclouds all the time, even though dataloader shuffled them.

To solve it, I did the following: points sampling and normalization happen before training, it's a static transformation. Noise application happens during training. This way, the models sees very similar yet different pointclouds.

The result: the loss is not decreasing too fast,

Weights Initialization

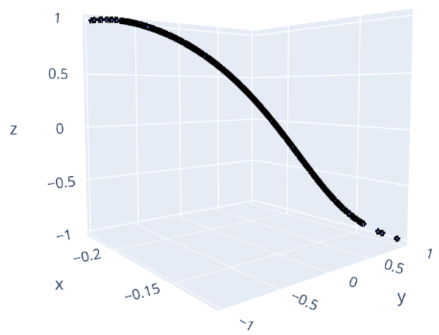
I use Xavier weights initialization, as mentioned here. I didn't notice much difference, probably because my model generally doesn't perform well.

`Conv1d`

Encoder part has `Conv1d` to increase the depth. For example, initially a pointcloud has shape `[3, 1024]`, I treat `3` as a number of channels and use `Conv1d` to increase it up to 1024. Nevertheless, decoder part consisted of only `Linear` layers. It increased number of neurons from `latent_space_dim` to `1024 * 3` which was then reshaped to `[3, 1024]`.

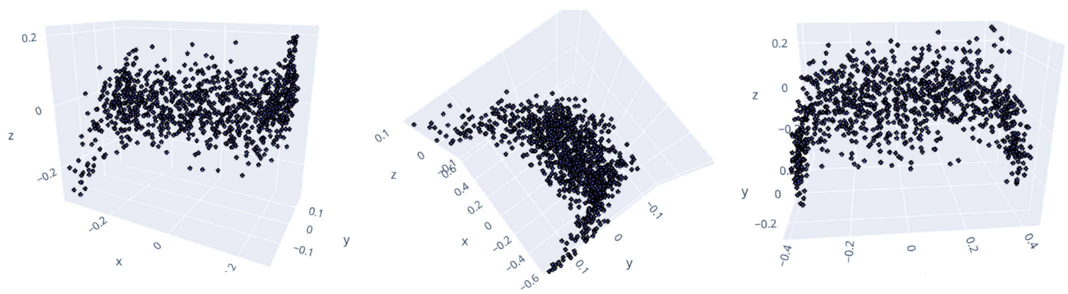
To increase number of dimensions

`Conv1d` could take `[1, 1024]` and expend the number of channels from 1 to 3 and output `[3, 1024]`. When I tried adding `TransposeConv1d`, I got this:



To increase number of channels

`Conv1d` takes shape `[1, 100]` where 100 is latent space shape. I turn 100 into 1024 using `Conv1d` and get shape `[1, 1024]`. Then reshape it to `[1024]` and pass to `Linear` layer to get `[1024*3]` and then reshape it to `[3, 1024]` - 1024 points, 3 dimensions. The results are (3 different ones):



So, my guess is that the model learns some features (length of bed, for example).

Activation Function

ReLU gives better results comparing to SELU.

Autoencoder

Current architecture is (with `Conv1d` in decoder to increase number of points):

```
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=100, bias=True)
    )
    (mu_fc): Linear(in_features=100, out_features=100, bias=True)
    (log_var_fc): Linear(in_features=100, out_features=100,
bias=True)
  )
  (decoder): PointDecoder(
    (conv): Sequential(
      (0): Conv1d(100, 128, kernel_size=(1,), stride=(1,))
      (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): SELU()
      (3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
      (4): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): SELU()
    )
    (dense_layers): Sequential(
      (0): Linear(in_features=1024, out_features=3072, bias=True)
      (1): Dropout(p=0.3, inplace=False)
      (2): Tanh()
    )
  )
```

```
)
```

Variational Autoencoder

This time I didn't focus much on adding *varionality*. It can be simply added by uncommenting using `reparametrize` method:

```
class PointNetAE(nn.Module):
    def __init__(self, num_points=1024, z_dim=100):
        super(PointNetAE, self).__init__()
        self.num_points = num_points

        self.encoder = PointEncoder(num_points, z_dim=z_dim)
        self.decoder = PointDecoder(num_points, z_dim=z_dim)

    def reparameterize(self, mu, log_var):
        std = torch.exp(log_var / 2)
        eps = torch.randn_like(std)
        return mu + std * eps

    def forward(self, x):
        x, mu, logvar = self.encoder(x)
        # uncomment the line below to turn AE into VAE
        # x = self.reparameterize(mu, logvar)
        x = self.decoder(x)
        return x
```

Training

Now, no matter with which decoder I train the autoencoder, loss gets stuck at ~120 and even if train loss goes a bit down, valid loss goes up. I use Adam with lr=0.001.

Expectations

I do not expect this project to actually achieve its initial goal - it's well-known that working with pointclouds is difficult. In every report I write everything new I've learned, which is helpful for my thesis.

Links

[GitHub repository]

[Folder with everything project-related]

What’s next

1. Try to do it for 2 or 3 classes
    1. with the model that overfits (previous report)
    2. with the model from this report
2. Compose all the information into one report
3. Add comments for classes and functions for better navigation