CS 726- Advanced ML

Assingment-1 Report

200100105 – Niteesh Singh

20D100007—Rushi Chavda

We will be using Earth Movers Distance(emd) for all the further analysis of the performance of the model at various settings and conditions. We will consider the best value of "test_emd" of the three runs as the final metric value.

We used learnable time embedding by passing the time step(t) of x_t to a small 2-layer NN which generates a t_w = 16-dimension output, which We then concatenated with respective x_t to get an n_dim + t_w = 19-dimension input for each training example (x_t^i) after time embedding.

```
t_w = 16
self.time_embed = torch.nn.Sequential(torch.nn.Linear(1, 16),
                                      torch.nn.Tanh(),
                                      torch.nn.Linear(16, t_w),
                                      torch.nn.Tanh())
```

Fig. time embedding model used

1.Changing the Number of Training Steps:

Current model setting

- Model structure is:
  - w = 256

```
self.model = torch.nn.Sequential(
    torch.nn.Linear(t_w+3, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 2*w),
    torch.nn.Tanh(),
    torch.nn.Linear(2*w, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 3)
)
```

- Hyperparameters:
  - lbeta: 1.0e-05
  - n_dim: 3
  - n_steps: 50
  - ubeta: 0.0128
  - batch_size: 1024
  - learning_rate: 0.001

Number of Epochs = 500 (n_epochs=500)

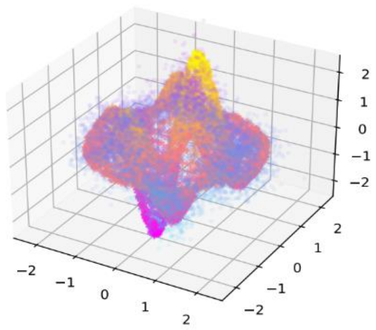
Results-

test_emd: 65.17553504955823

train_emd: 70.40486521231745

test_nll: 2.6779849529266357

train_nll: 2.670085906982422



Number of Epochs = 1000 (n_epochs=1000)

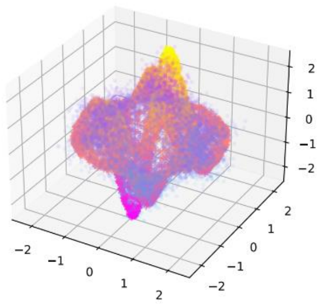
Results

test_emd: 66.36065731543384

train_emd: 66.31490655981473

test_nll: 2.6448776721954346

train_nll: 2.6374194622039795



Number of Epochs = 2000 (n_epochs=2000)

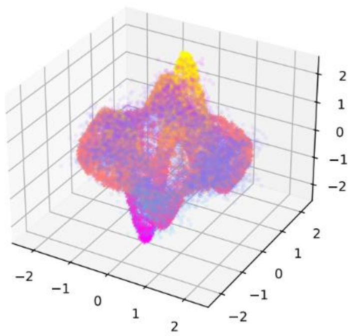
Results

test_emd: 63.07423301135008

train_emd: 62.0009674093369

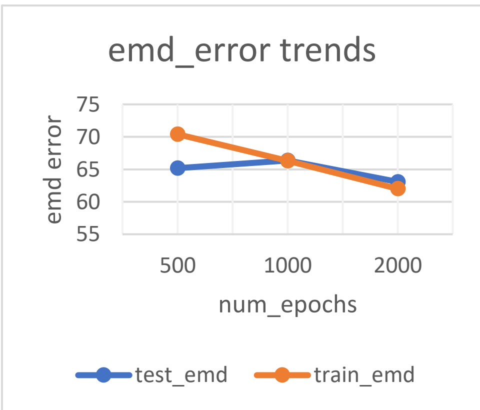
test_nll: 2.641606092453003

train_nll: 2.63297176361084



Analysis

- As the number of epochs increases there is very small change in test error, but it has decreased for large value so an indication that trying large value for num_epochs can be beneficial.
- train error decreases as num_epochs are increased it may be due to overfitting.
- So maybe we should some optimal number of epochs between 500-1000 and it would perform better because sample generated for higher num_epochs values are better as per visualisation.
- So, lets choose 600 as optimal value for n_epoch.



2. Model complexity:

Current model setting

- W = 256
- Hyperparameters:
    - lbeta: 1.0e-05
    - n_dim: 3
    - n_steps: 50
    - ubeta: 0.0128
    - batch_size: 1024
    - learning_rate: 0.001
    - n_epochs: 600

Simple Model, 72K trainable parameters

```
self.model = torch.nn.Sequential(
    torch.nn.Linear(t_w+3, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 3)
)
```

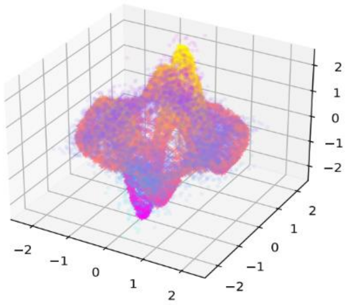
Results

test_emd: 63.30597077579907

train_emd: 64.55693641071107

test_nll: 2.654146671295166

train_nll: 2.6470699310302734



Complex Model, 1.3M trainable parameters

```
self.model = torch.nn.Sequential(
    torch.nn.Linear(t_w+3, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 2*w),
    torch.nn.Tanh(),
    torch.nn.Linear(2*w, 4*w),#
    torch.nn.Tanh(),#
    torch.nn.Linear(4*w, 2*w),#
    torch.nn.Tanh(),#
    torch.nn.Linear(2*w, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 3)
)
```

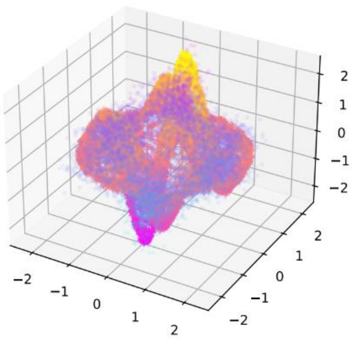
Results

test_emd: 70.31164168610466

train_emd: 62.754993318713986

test_nll: 2.6775283813476562

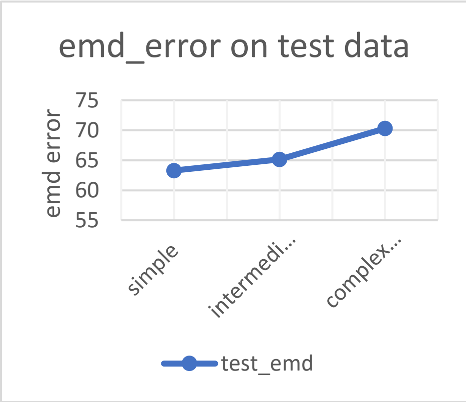
train_nll: 2.6704649925231934



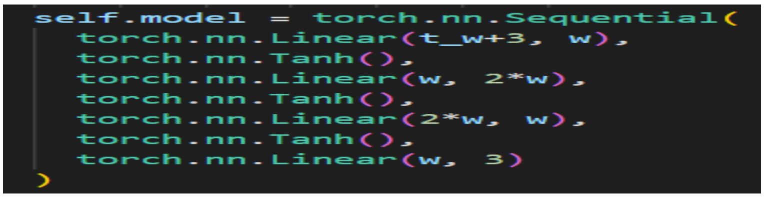
NOTE: the third intermediate model is same as we trained on previous part of the assingment, and the number of epochs are similar so have not trained it again

Analysis

- As the model gets complex training time increases.
- training error has increased for very complex model.
- It seems that a very simple model works fine for this toy dataset.
- We will choose an intermediate model, the model in which we did number of epoch testing because it seems like a good model with enough parameters and optimal training time and memory requirements.



So, our final model is – (w=256)



And num_epochs = 600

3. Number of diffusion steps (T):

For 3d_sin_5_5 data

Current model setting

- W = 256
- Hyperparameters:
  - lbeta: 1.0e-05
  - n_dim: 3
  - ubeta: 0.0128
  - batch_size: 1024
  - learning_rate: 0.001
  - n_epochs: 600
  - dataset used = 3d_sin_5_5 data

Number of Steps = 10, n_steps = 10

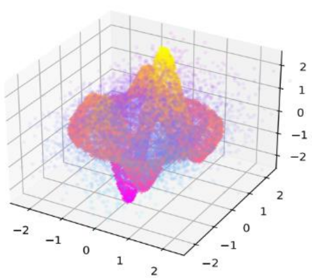
Results

test_emd: 83.30035817586062

train_emd: 81.41897810190828

test_nll: 2.9109654426574707

train_nll: 2.8995320796966553



Number of Steps = 50, n_steps = 50

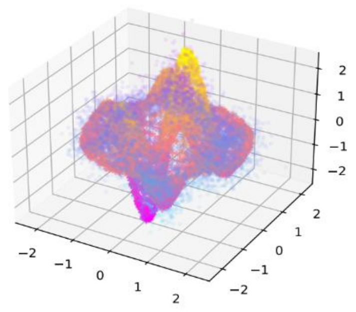
Results-

test_emd: 65.17553504955823

train_emd: 70.40486521231745

test_nll: 2.6779849529266357

train_nll: 2.670085906982422



Number of Steps = 100, n_steps = 100

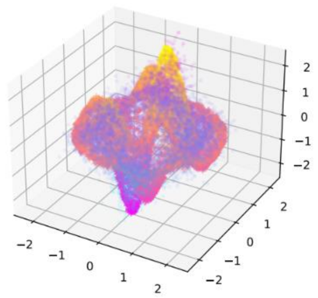
Results

test_emd: 62.59729977226936

train_emd: 64.6123901338262

test_nll: 2.6689274311065674

train_nll: 2.6577916145324707



Number of Steps = 150, n_steps = 150

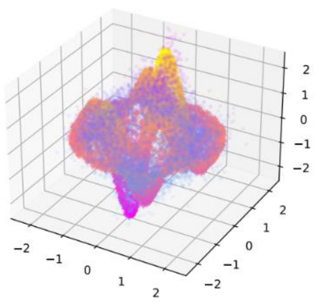
Results

test_emd: 64.7138290425177

train_emd: 66.87266873899209

test_nll: 2.6902542114257812

train_nll: 2.6812427043914795



Number of Steps = 200, n_steps = 200

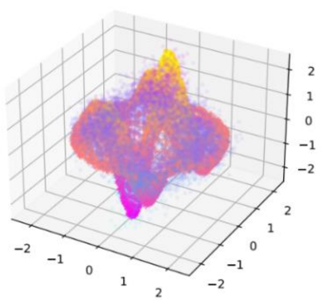
Results

test_emd: 69.04824575353155

train_emd: 69.0003225412479

test_nll: 2.680298089981079

train_nll: 2.670194387435913



for helix_3D data

Current model setting

- W = 256
- Hyperparameters:
    - lbeta: 1.0e-05
    - n_dim: 3

- ubeta: 0.0128
- batch_size: 1024
- learning_rate: 0.001
- n_epochs: 600

Number of Steps = 10, n_steps = 10

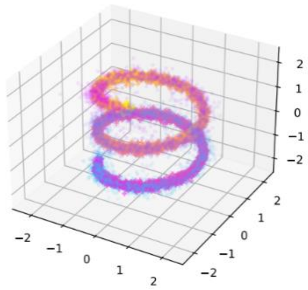
Results

test_emd: 58.99251491490972

train_emd: 48.61157009063477

test_nll: 2.352220058441162

train_nll: 2.346301317214966



Number of Steps = 50, n_steps = 50

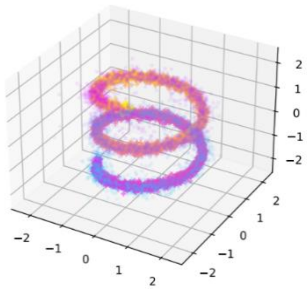
Results

test_emd: 51.35311040806201

train_emd: 70.57267437383348

test_nll: 2.3485875129699707

train_nll: 2.3438773155212402



Number of Steps = 100, n_steps = 100

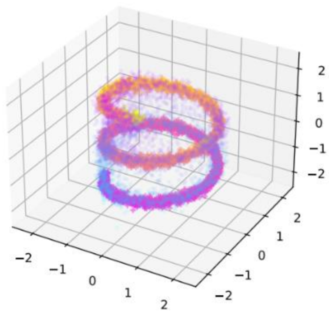
Results

test_emd: 51.60572010533156

train_emd: 50.26444759378028

test_nll: 2.3474135398864746

train_nll: 2.3469741344451904



Number of Steps = 150, n_steps = 150

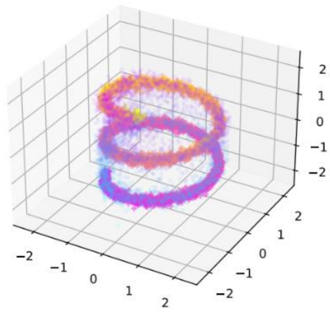
Results

test_emd: 53.12005686204001

train_emd: 63.921479221190125

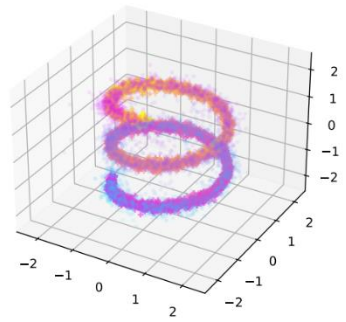
test_nll: 2.3685905933380127

train_nll: 2.3659677505493164



Number of Steps = 200, n_steps = 200

Results



test_emd: 46.17424568125869

train_emd: 50.38415121867081

test_nll: 2.3596997261047363

train_nll: 2.357518196105957

Analysis

- Here we have mentioned the best results out of the three evaluation, for all the different case.
- By overlooking at the data we can infer that by increasing the steps (T) we can get better results i.e., smaller value for EMD
- We can observe that after 100 steps the change in EMD is not significance

4. Number of diffusion steps (T):

For 3d_sin_5_5 data

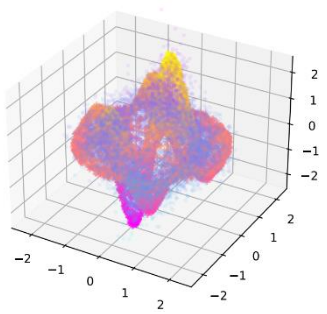
Current model setting

- W = 256
- Hyperparameters:
    - lbeta: 1.0e-05
    - n_dim: 3
    - n_steps: 150
    - ubeta: 0.0128
    - batch_size: 1024
    - learning_rate: 0.001
    - n_epochs: 600

"Linear" Noise schedule

Beta values are linearly varying with t

Results



test_emd: 68.95116409595362

train_emd: 71.0804771180983

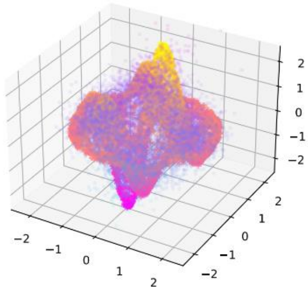
test_nll: 2.6895651817321777

train_nll: 2.680345058441162

“Constant” Noise schedule

Beta values are equal to ubeta value for all t.

Results



test_emd: 63.72570935469814

train_emd: 74.0343293062074

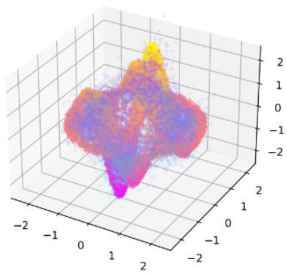
test_nll: 2.7506325244903564

train_nll: 2.739440679550171

“Quadratic” Noise schedule

Beta values are following a quadratic function over t and range is [lbeta, ubeta].

Results



test_emd: 67.027948243499

train_emd: 70.11542925460446

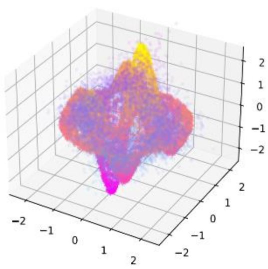
test_nll: 2.677572250366211

train_nll: 2.668335437774658

“warmup10” Noise schedule

Beta values ranges linearly from [lbeta to ubeta] in first 10% of total timesteps and later all values of betas are equal to ubeta. {on graph- linear function followed by const. value function}

Results



test_emd: 71.02178100826868

train_emd: 86.69617083708691

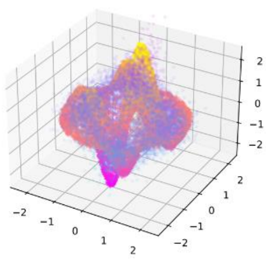
test_nll: 2.724071979522705

train_nll: 2.7135508060455322

“warmup50” Noise schedule

Beta values ranges linearly from [lbeta to ubeta] in first 50% of total timesteps and later all values of betas are equal to ubeta. {on graph- linear function followed by const. value function}

Results



test_emd: 66.07125496063745

train_emd: 76.25061033049296

test_nll: 2.7245938777923584

train_nll: 2.7167930603027344

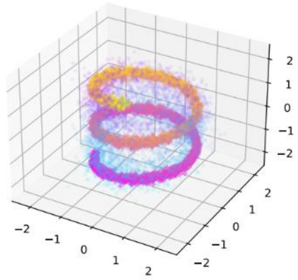
For helix_3D data

Current model setting

- W = 256
- Hyperparameters:
  - lbeta: 1.0e-05
  - n_dim: 3
  - n_steps: 150
  - ubeta: 0.0128
  - batch_size: 1024
  - learning_rate: 0.001
  - n_epochs: 600

"Linear" Noise schedule

Results



test_emd: 59.00353418889661

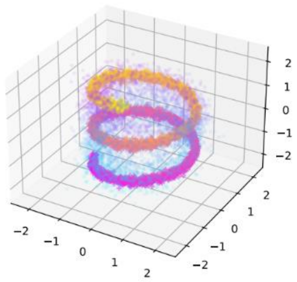
train_emd: 74.9186193992157

test_nll: 2.3739233016967773

train_nll: 2.370929002761841

"Constant" Noise schedule

Results



test_emd: 70.05747486392498

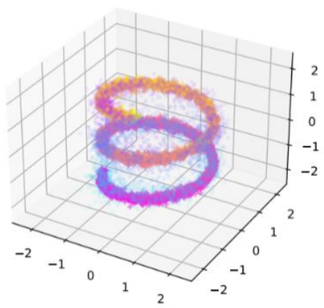
train_emd: 64.9508319559166

test_nll: 2.646155595779419

train_nll: 2.644508123397827

"Quadratic" Noise schedule

Results



test_emd: 59.34619851891867

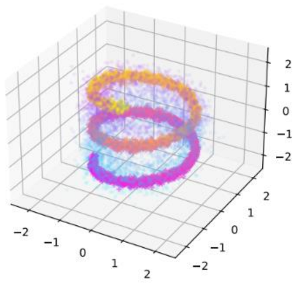
train_emd: 61.26329202036436

test_nll: 2.3793678283691406

train_nll: 2.3778738975524902

"warmup10" Noise schedule

Results



test_emd: 69.262612540213

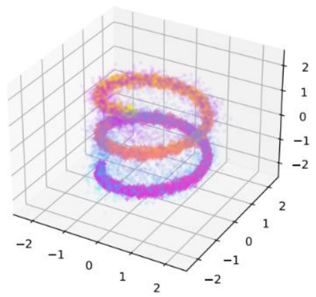
train_emd: 64.53478288896034

test_nll: 2.568970203399658

train_nll: 2.5658304691314697

“warmup50” Noise schedule

Results



test_emd: 63.84509899787168

train_emd: 63.27920885475712

test_nll: 2.4459991455078125

train_nll: 2.4439995288848877

Analysis

- In both dataset the linear scheduling seems to work fine.
- Constant noise factor destroys the information very quickly, and seems to not work very well.
- The warmup50 noise scheduling also works fine and in 3d_sin_5_5 data its produces better result.

Final Results/Conclusion

- Our final model is-

```
self.model = torch.nn.Sequential(
    torch.nn.Linear(t_w+3, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 2*w),
    torch.nn.Tanh(),
    torch.nn.Linear(2*w, w),
    torch.nn.Tanh(),
    torch.nn.Linear(w, 3)
)
```

- Optimal number of epochs – 600
- Number of Steps(T) = 150
- Optimal Noise Scheduler = Linear and warmup50

--- Thank You ---