AutoEncoderBySize(alpha=5)

December 8, 2021

1 Autoencoder Performance by Size

In this notebook, we investigate the performance of a basic autoencoder model iwhen neighbors become random in the high-dimensional space according to the data size. We start by loading the necessary libraries.

[1]:
```
# Handling arrays
import numpy as np

# Handling data frames
import pandas as pd

# neural networks in Python
import torch
from torch import nn

# plotting
import matplotlib.pyplot as plt

%matplotlib inline
```

We define our experiment parameters.

[2]:
```
ntimes = 100 # number of noise replicates to investigate dimred performance
npoints = [25, 50, 100] # number of points in our ground truth data set to be␣
↪investigated
maxdim = 10000 # maximal dimension of the data set to be investigated
dims = np.round(np.exp(np.linspace(np.log(2), np.log(maxdim), num=10))).
↪astype("int") # dimensions to study
a = 1.25 # magnitude of noise: per dimension we sample noise uniformly from␣
↪[-a, a]
alpha = 5 # factor controling the growth rate of the ground truth diamete
```

We construct the ground truth data sets according to the various growth rates.

[3]:
```
datasets = []
for idx, points in enumerate(npoints):
    t = np.linspace(0, 1, num=points)
```

```
    factor = np.ones(maxdim) if alpha == np.inf else (np.arange(maxdim) +␣
↪1)**(-1 / alpha)
    datasets.append(np.transpose(np.tile(t, (maxdim, 1))) * np.tile(factor,␣
↪(points, 1)))
```

We define an autoencoder model.

[4]:
```
class autoencoder(nn.Module):

    def init_weights(self, m):
        if isinstance(m, nn.Linear):
            nn.init.xavier_uniform_(m.weight, gain=1.0)
            nn.init.zeros_(m.bias)

    def __init__(self, input_dim, encoding_dim):
        super(autoencoder, self).__init__()
        self.input_dim = input_dim
        self.encoding_dim = encoding_dim
        self.encoder = nn.Sequential(
            nn.Linear(self.input_dim, 24),
            nn.Tanh(),
            nn.Linear(24, 6),
            nn.Tanh(),
            nn.Linear(6, self.encoding_dim),
            nn.Tanh())
        self.decoder = nn.Sequential(
            nn.Linear(self.encoding_dim, 6),
            nn.Tanh(),
            nn.Tanh(),
            nn.Linear(6, 24),
            nn.Linear(24, self.input_dim))

        self.encoder.apply(self.init_weights)
        self.decoder.apply(self.init_weights)

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x


def autoencode(X, encoding_dim, num_epochs=2000, learning_rate=1e-3, eps=1e-07):
    X = torch.tensor(X).type(torch.float)
    model = autoencoder(input_dim=X.shape[1], encoding_dim=encoding_dim)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, eps=eps)
```

```
    for epoch in range(num_epochs):
        output = model(X)
        loss = criterion(output, X)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    Y = model.encoder(X).detach().numpy()

    return(Y)
```

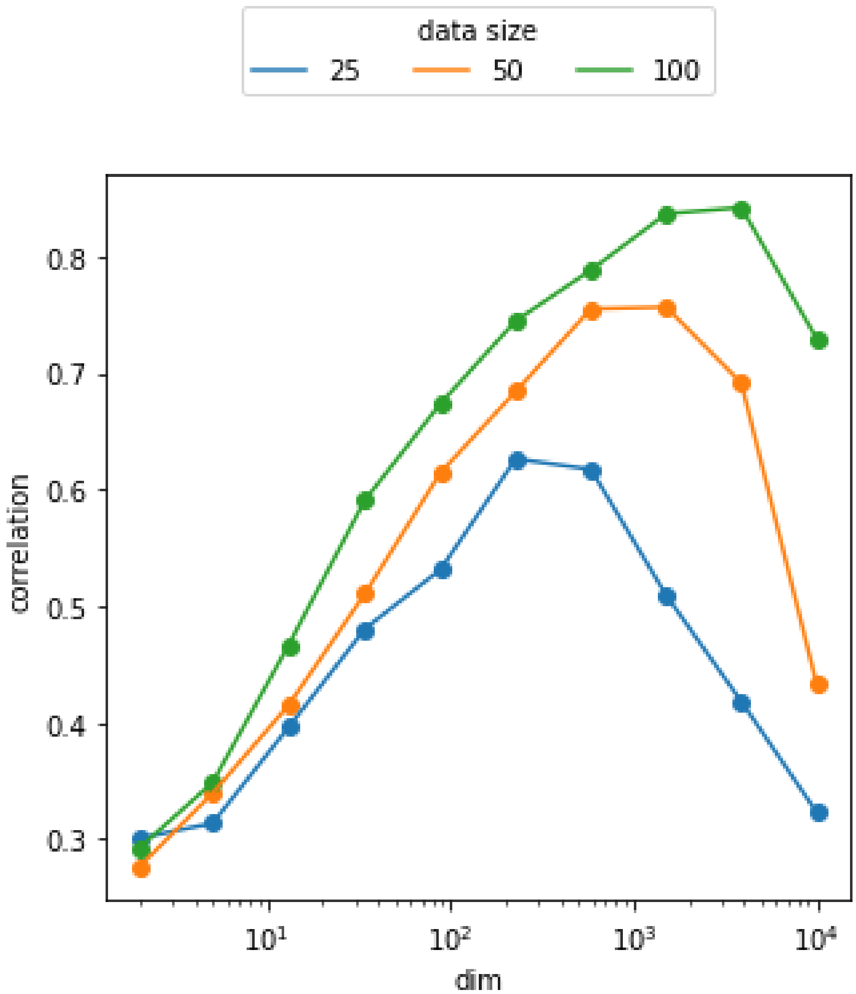
We measure the autoencoder performance by dimensionality and data size.

```
[5]: cor_auto = np.zeros([len(dims), len(npoints)])

for idx in range(ntimes):

    print("progress: " + str(round(100 * idx / ntimes, 2)).ljust(5, "0") + "%",␣
 ↪end="\r")

    N = a * (2 * np.random.rand(max(npoints), maxdim) - 1)

    for np_idx, points in enumerate(npoints):

            XN = datasets[np_idx] + a * N[:datasets[np_idx].shape[0],]

            for dim_idx, dim in enumerate(dims):

                Y = autoencode(XN[:,:dim], 1)
                cor = np.max([np.corrcoef(Y[:,0], datasets[np_idx][:,0])[0, 1],
                              np.corrcoef(np.flip(Y[:,0], axis=0),␣
 ↪datasets[np_idx][:,0])[0, 1]])
                cor_auto[dim_idx, np_idx] += cor

cor_auto /= ntimes

print("progress: 100.0%", end="\r")


fig, ax = plt.subplots(figsize=(5, 5))
ax.set_xlabel("dim")
ax.set_ylabel("correlation")
ax.set_xscale("log")

for idx, points in enumerate(npoints):

    ax.plot(dims, cor_auto[:,idx], label=points)
```

```
    ax.scatter(dims, cor_auto[:,idx])

ax.legend(title="data size", loc="upper center", ncol=3, bbox_to_anchor=(0.5, 1.
↪25))
```

```
progress: 100.0%
```

[5]: <matplotlib.legend.Legend at 0x7f123cda45e0>



We modify the style of the results and save as csv for plotting in R.

[6]:
```
cor_auto_df = np.zeros([cor_auto.shape[0] * cor_auto.shape[1], 3])
idx = 0
for idx1, points in enumerate(npoints):
    for idx2, dim in enumerate(dims):
        cor_auto_df[idx,:] = [points, dim, cor_auto[idx2, idx1]]
```

```
        idx += 1

cor_auto_df = pd.DataFrame(cor_auto_df)
cor_auto_df.columns = ["size", "dim", "cor"]
cor_auto_df.to_csv("../Results/Size/AUTO_alpha5.csv")
```

[ ]:

```

```