

Assignment 4

Wanyi Su
#301445656

Q1 SVM

1. choose #4

- ① The problem defines a "Soft-margin linear SVM", and no non-linearity is added, so the hyperplane should be linear. So we limit choice to 3, 4.
- ② Although this is a soft margin SVM, penalty hyperparameter C is 0.02, which is small. This means the SVM still values a large margin with a little tolerance of misclassification. relatively

2. choose #3

- ① The problem defines a "Soft-margin linear SVM", and no non-linearity is added, so the hyperplane should be linear. So we limit choice to 3, 4.
- ② This is a soft-margin SVM with a large penalty hyperparameter C of 20. This means the SVM tolerates some level of misclassification with a trade-off of sacrificing maximum margin. So compared with #4, this SVM is more generalized and has a narrower margin.

3. choose #5

With a polynomial kernel of 2-degree, data points are transformed into a higher-dimensional feature space to be separated.

4. choose #6.

With a RBF kernel of gamma values 5, similarity radius is relatively small, which means data points need to be close to each other to be in the same class. A small noise may affect a data point's classification. This means SVM can follow too much on noises. So the shape of the decision boundary can be not so smooth.

5. choose #1.

With a RBF kernel of gamma values at $\frac{1}{5}$, similarity radius is relatively larger. More data points can be classified together, and the areas on two sides of boundary are larger to incorporate more data points than #6. This SVM is more generalized and has smoother curve.

Q2

#1

Implement the architecture of the encoder and decoder and print out the reconstruction losses on the train and validation sets.

Also use the scatter plot function in autoencoder_starter.py to plot the 2D bottleneck feature representation as a scatter plot, where different classes are represented by dots of different colours. Include a screenshot of your code that implements the architecture and the generated scatter plot in your PDF submission.

The screenshot shows a Google Colab notebook titled "Autoencoder_sample.ipynb". The code cell [4] contains the following Python code:

```
[4]: feature_size = 28 * 28
      # feature_shape = (mini_batch, (1, 28, 28)). # grayscale: 1, RGB: 3
      feature_shape = (1, 28, 28)
      label_counts = 10
```

Below the code, there is a note: "Define your architecture here." and a warning: "The Autoencoder class has several important functions unimplemented. You are required to implement the two sub-classes of Encoder and Decoder, i.e, the architecture and forward function of the encoder and decoder."

The screenshot shows the continuation of the Autoencoder implementation in the same Google Colab notebook. The code cell [4] now includes the definition of the Autoencoder class and its Encoder sub-class:

```
[4]: class Autoencoder(nn.Module):
    def __init__(self, dim_latent_representation=2):
        super(Autoencoder, self).__init__()
        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder_layers = nn.Sequential(
                    # nn.Flatten(), # flatten to one-dimension
                    nn.Linear(in_features=feature_size, out_features=output_size)
                )
            def forward(self, x):
                # needs your implementation
                # flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten()
                f = nn.Flatten()
                fx = f(x)
                z = self.encoder_layers(fx)
                return z
```

The image shows two screenshots of a Google Colab notebook titled "Autoencoder_sample.ipynb".

Top Screenshot:

```

class Decoder(nn.Module):
    def __init__(self, input_size=2):
        super(Decoder, self).__init__()
        # needs your implementation
        self.decoder_layers = nn.Sequential(
            nn.Linear(
                in_features=input_size, out_features=feature_size),
            nn.Sigmoid()
        )

    def forward(self, z):
        # needs your implementation
        # reconstructed_no_sigmoid = self.decoder_output_layer(z)
        # reconstructed_no_reshape = self.decoder_output_layer_sigmoid(reconstructed_no_sigmoid)
        reconstructedz = self.decoder_layers(z)
        r = nn.Unflatten(1, feature_shape)
        reconstructed = r(reconstructedz)
        return reconstructed

    self.encoder = Encoder(output_size=dim_latent_representation)
    self.decoder = Decoder(input_size=dim_latent_representation)

```

Bottom Screenshot:

```

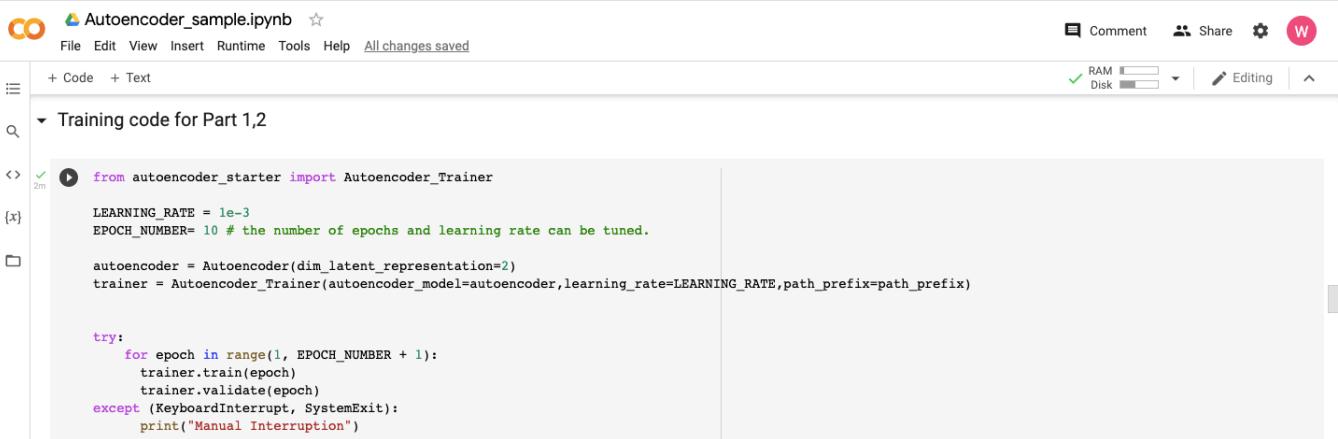
# Implement this function for the DAE model
# def add_noise(self, x, noise_type):
#     if noise_type=='Gaussian':
#         # return (x with Gaussian noise)
#     elif noise_type=='Dropout':
#         # return (x with Dropout noise)

# Implement this function for the VAE model
# def reparameterise(self, mu, logvar):
#     if self.training:
#         # return reparametrized mu
#     else:
#         return mu

def forward(self,x):
    # This function should be modified for the DAE and VAE
    x = self.encoder(x)
    x = self.decoder(x)
    # for the VAE forward function should also return mu and logvar
    return x

```

Training code:



```

from autoencoder_starter import Autoencoder_Trainer

LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

autoencoder = Autoencoder(dim_latent_representation=2)
trainer = Autoencoder_Trainer(autoencoder_model=autoencoder,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")

```

Losses:

I have tried LEARNING RATE=[1e-4, 1e-3, 1e-2] and EPOCH NUMBER=[10, 20], it turns out that the set of 1e-3 and 10 works relatively better to get training and validation loss around 0.75.



```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested cpuset_checked)
  100%|██████████| 1875/1875 [00:14<00:00, 129.24it/s]==> Epoch: 1 Average loss: 0.9197

  100%|██████████| 313/313 [00:02<00:00, 146.28it/s]==> Val set loss (reconstruction error) : 0.8240
  100%|██████████| 1875/1875 [00:14<00:00, 130.24it/s]==> Epoch: 2 Average loss: 0.8048

  100%|██████████| 313/313 [00:02<00:00, 141.34it/s]==> Val set loss (reconstruction error) : 0.7876

  100%|██████████| 1875/1875 [00:14<00:00, 130.38it/s]==> Epoch: 3 Average loss: 0.7810

  100%|██████████| 313/313 [00:02<00:00, 150.62it/s]==> Val set loss (reconstruction error) : 0.7711

  100%|██████████| 1875/1875 [00:14<00:00, 129.81it/s]==> Epoch: 4 Average loss: 0.7679

  100%|██████████| 313/313 [00:02<00:00, 145.08it/s]==> Val set loss (reconstruction error) : 0.7596

  100%|██████████| 1875/1875 [00:14<00:00, 129.64it/s]==> Epoch: 5 Average loss: 0.7568

  100%|██████████| 313/313 [00:02<00:00, 145.72it/s]==> Val set loss (reconstruction error) : 0.7568

  100%|██████████| 1875/1875 [00:14<00:00, 128.30it/s]==> Epoch: 6 Average loss: 0.7586

  100%|██████████| 313/313 [00:02<00:00, 147.47it/s]==> Val set loss (reconstruction error) : 0.7545

  100%|██████████| 1875/1875 [00:14<00:00, 129.63it/s]==> Epoch: 7 Average loss: 0.7563

  100%|██████████| 313/313 [00:02<00:00, 145.79it/s]==> Val set loss (reconstruction error) : 0.7539

  100%|██████████| 1875/1875 [00:14<00:00, 127.50it/s]==> Epoch: 8 Average loss: 0.7547

  100%|██████████| 313/313 [00:02<00:00, 145.24it/s]==> Val set loss (reconstruction error) : 0.7518

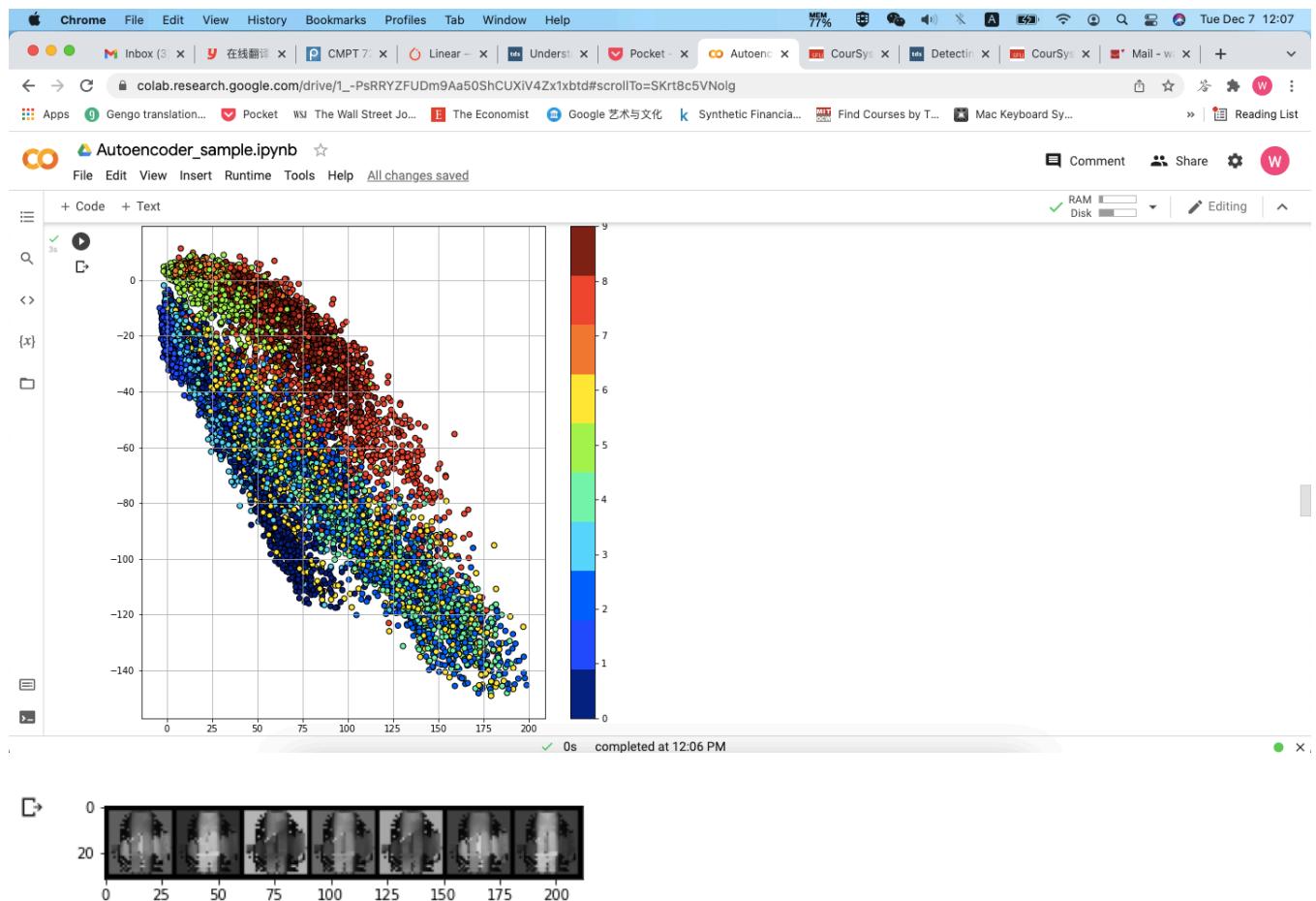
  100%|██████████| 1875/1875 [00:14<00:00, 130.36it/s]==> Epoch: 9 Average loss: 0.7539

  100%|██████████| 313/313 [00:02<00:00, 148.37it/s]==> Val set loss (reconstruction error) : 0.7517

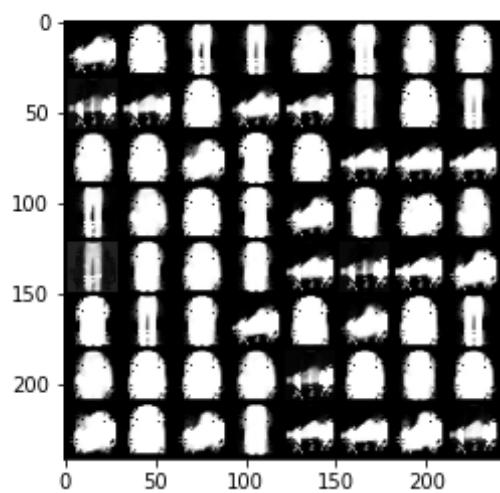
  100%|██████████| 1875/1875 [00:14<00:00, 129.12it/s]==> Epoch: 10 Average loss: 0.7536

  100%|██████████| 313/313 [00:02<00:00, 146.21it/s]==> Val set loss (reconstruction error) : 0.7514

```



Reconstructed images



#2

Print out the reconstruction losses on the train and validation sets and generate a scatter plot of the same form as in part (1).

Include a screenshot of the code that implements the architecture and the generated scatter plot in the PDF submission.



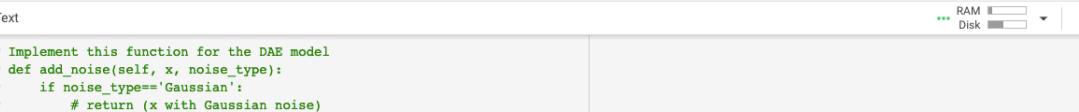
```
class Autoencoder(nn.Module):
    def __init__(self, dim_latent_representation=2):
        super(Autoencoder, self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.encoder_layers = nn.Sequential(
                    nn.Linear(in_features=feature_size, out_features=1024),
                    nn.ReLU(),
                    nn.Linear(in_features=1024, out_features=output_size)
                )

            def forward(self, x):
                # needs your implementation
                # flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten()
                f = nn.Flatten() # flatten to one-dimension
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** "Autoencoder_sample.ipynb" with a star icon.
- Header:** File, Edit, View, Insert, Runtime, Tools, Help, Saving....
- Toolbar:** Comment, Share, Settings, and a "W" icon.
- Code Cell:** Contains Python code for an Autoencoder. The code defines a `Decoder` class that takes `input_size=2` and uses `nn.Sequential` to define its layers: two `Linear` layers with `ReLU` activation and a `Sigmoid` layer at the end. It also defines a `forward` method that reconstructs the input `z` into a 1D tensor `r` using `Unflatten`. The notebook is currently at cell 0s.
- Cell Controls:** Includes a play button, a green checkmark, and a search icon.
- Right Panel:** RAM and Disk usage monitoring, an "Editing" button, and a scroll bar.



The screenshot shows a Jupyter Notebook interface with the following details:

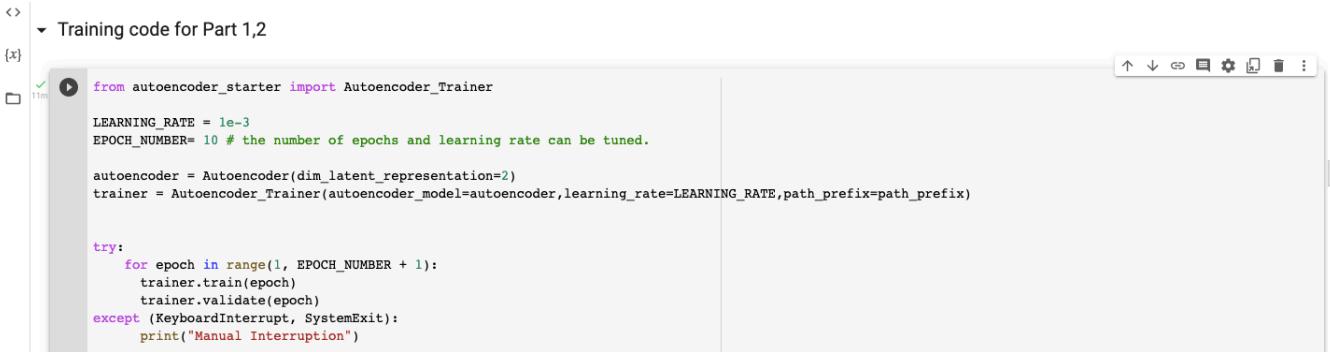
- Title Bar:** "Autoencoder_sample.ipynb" with a star icon.
- Header:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved.
- Toolbar:** Comment, Share, Settings, and a profile icon.
- Code Area:** The code block contains Python code for an Autoencoder. It includes functions for adding noise (add_noise), reparameterizing (reparameterise), and forward passes (forward). The code uses docstrings and comments to describe its purpose.
- Code Editor:** Shows code completion suggestions and a preview of the code.
- RAM and Disk Status:** RAM and Disk usage indicators.

```
# Implement this function for the DAE model
# def add_noise(self, x, noise_type):
#     if noise_type=='Gaussian':
#         # return (x with Gaussian noise)
#     elif noise_type=='Dropout':
#         # return (x with Dropout noise)

# Implement this function for the VAE model
# def reparameterise(self, mu, logvar):
#     if self.training:
#         # return reparametrized mu
#     else:
#         return mu

def forward(self,x):
    # This function should be modified for the DAE and VAE
    x = self.encoder(x)
    x = self.decoder(x)
    # for the VAE forward function should also return mu and logvar
    return x
```

Training code:



```
<> ▾ Training code for Part 1,2
{x} 11m
  from autoencoder_starter import Autoencoder_Trainer

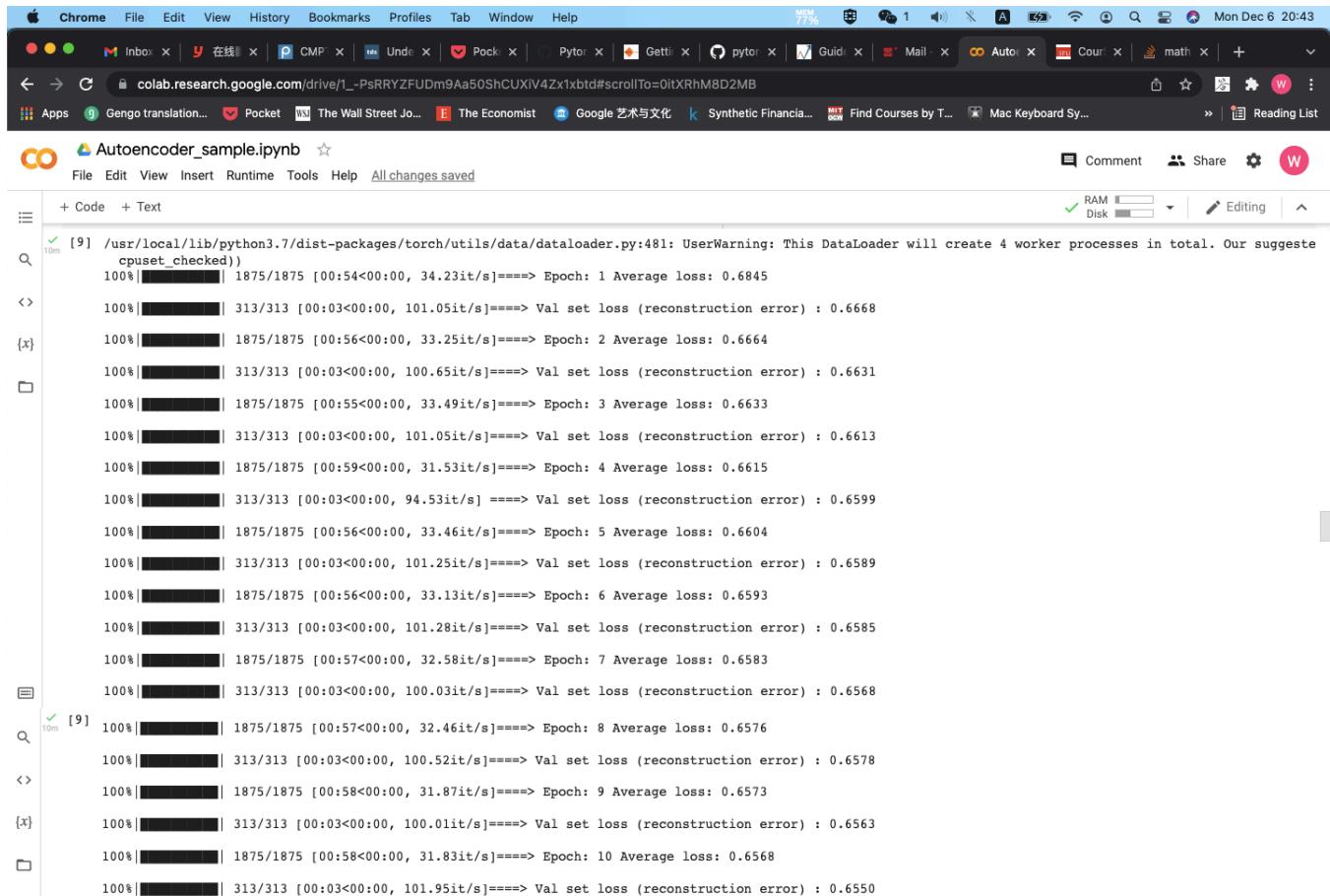
  LEARNING_RATE = 1e-3
  EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

  autoencoder = Autoencoder(dim_latent_representation=2)
  trainer = Autoencoder_Trainer(autoencoder_model=autoencoder,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

  try:
    for epoch in range(1, EPOCH_NUMBER + 1):
      trainer.train(epoch)
      trainer.validate(epoch)
  except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")
```

Losses:

After comparison of several sets of hyperparameters, it turns out that the set of 1e-3 and 20 works relatively better to get training and validation loss around 0.66.



Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

[9] /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested cpuset_checked()
100% [██████████] 1875/1875 [00:54<00:00, 34.23it/s]==> Epoch: 1 Average loss: 0.6845

100% [██████████] 313/313 [00:03<00:00, 101.05it/s]==> Val set loss (reconstruction error) : 0.6668

100% [██████████] 1875/1875 [00:56<00:00, 33.25it/s]==> Epoch: 2 Average loss: 0.6664

100% [██████████] 313/313 [00:03<00:00, 100.65it/s]==> Val set loss (reconstruction error) : 0.6631

100% [██████████] 1875/1875 [00:55<00:00, 33.49it/s]==> Epoch: 3 Average loss: 0.6633

100% [██████████] 313/313 [00:03<00:00, 101.05it/s]==> Val set loss (reconstruction error) : 0.6613

100% [██████████] 1875/1875 [00:59<00:00, 31.53it/s]==> Epoch: 4 Average loss: 0.6615

100% [██████████] 313/313 [00:03<00:00, 94.53it/s] ==> Val set loss (reconstruction error) : 0.6599

100% [██████████] 1875/1875 [00:56<00:00, 33.46it/s]==> Epoch: 5 Average loss: 0.6604

100% [██████████] 313/313 [00:03<00:00, 101.25it/s]==> Val set loss (reconstruction error) : 0.6589

100% [██████████] 1875/1875 [00:56<00:00, 33.13it/s]==> Epoch: 6 Average loss: 0.6593

100% [██████████] 313/313 [00:03<00:00, 101.28it/s]==> Val set loss (reconstruction error) : 0.6585

100% [██████████] 1875/1875 [00:57<00:00, 32.58it/s]==> Epoch: 7 Average loss: 0.6583

100% [██████████] 313/313 [00:03<00:00, 100.03it/s]==> Val set loss (reconstruction error) : 0.6568

[9] 100% [██████████] 1875/1875 [00:57<00:00, 32.46it/s]==> Epoch: 8 Average loss: 0.6576

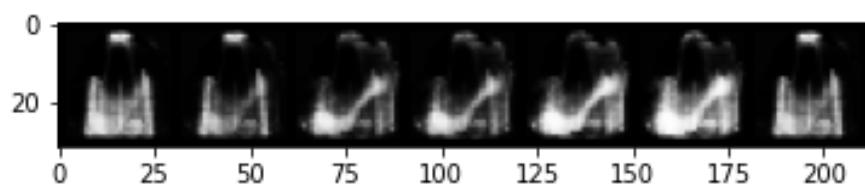
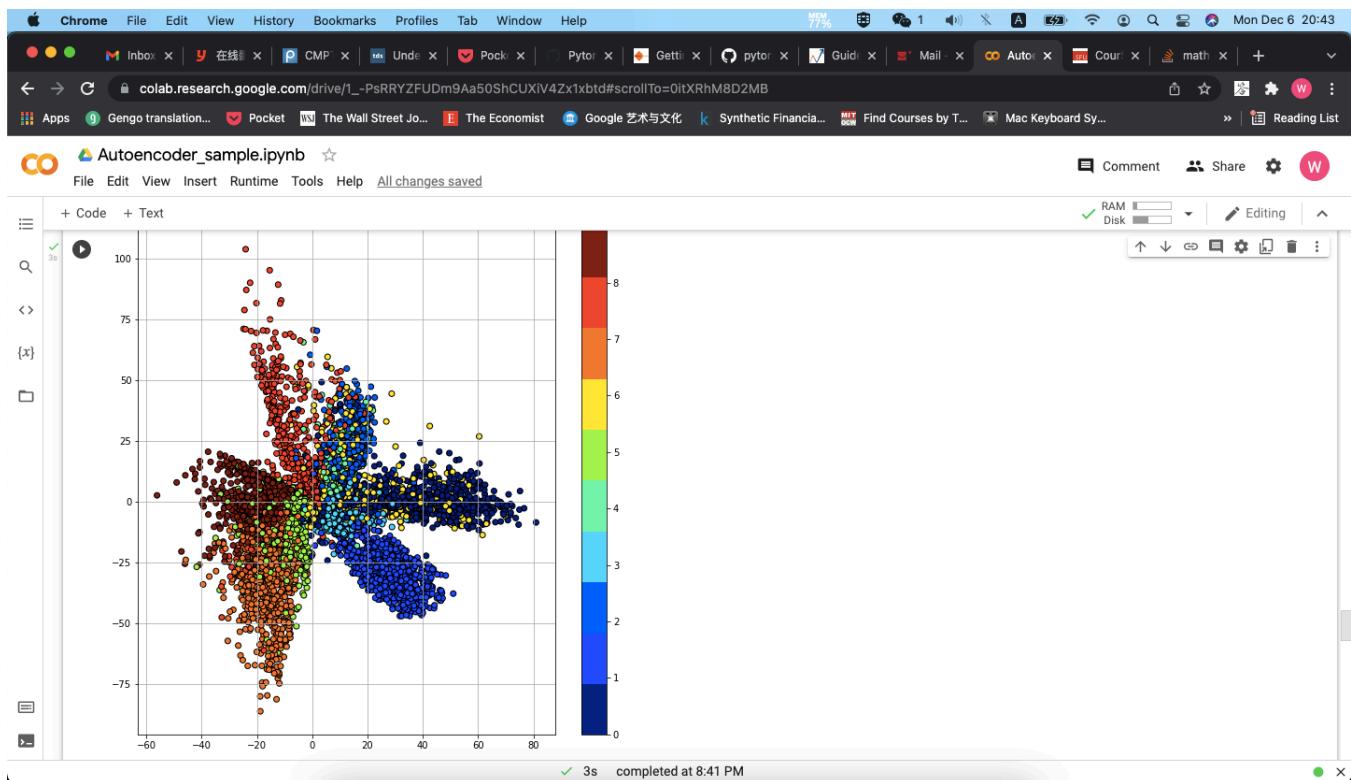
100% [██████████] 313/313 [00:03<00:00, 100.52it/s]==> Val set loss (reconstruction error) : 0.6578

100% [██████████] 1875/1875 [00:58<00:00, 31.87it/s]==> Epoch: 9 Average loss: 0.6573

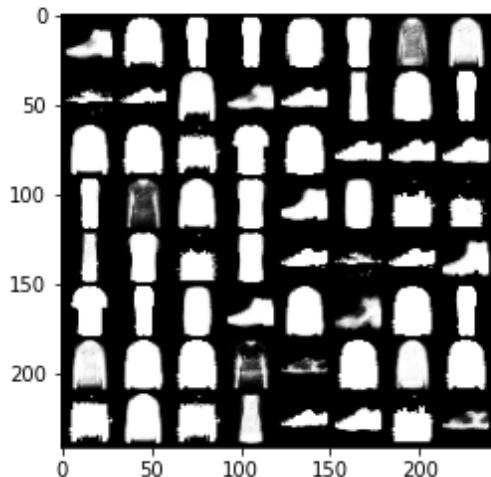
100% [██████████] 313/313 [00:03<00:00, 100.01it/s]==> Val set loss (reconstruction error) : 0.6563

100% [██████████] 1875/1875 [00:58<00:00, 31.83it/s]==> Epoch: 10 Average loss: 0.6568

100% [██████████] 313/313 [00:03<00:00, 101.95it/s]==> Val set loss (reconstruction error) : 0.6550



Reconstructed images



Describe how the plot differs from the one in part (1) and explain what this says about the architectures in this part and part (1). Why do you think the architecture in this part gave rise to the results shown in the scatter plot?

Points of different colors are separated more clearly and have longer between-groups distance with each other in part 2, and points of same colors gather more compactly than in part 1. Also, part 2's training and validation losses are relatively lower (around 0.65). This means part 2's architecture learns better and captures the images more accurately than part 1's architecture.

Why:

Part 1's architecture has only linear layer, while part 2's architecture has more layers and ReLU activation layer. Part 2's architecture has some non-linearity added in and is a more complex network, compared to part 1. More complex architecture with more layers enables the architecture to capture features accurately and learn better. If we only include linear activation functions in a neural network, the output will just be a linear transformation of the input, which may be not enough to form a universal function approximator.

#3

[autoencoder with no noise](#)

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share ⚙ W

+ Code + Text

RAM Disk Editing

Part 3 (no noise)

```
[25] class Autoencoder(nn.Module):
    def __init__(self, dim_latent_representation=2):
        # self.noise_type = noise_type
        super(Autoencoder, self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.layers = nn.Sequential(
                    nn.Linear(in_features=feature_size, out_features=output_size)
                )

            def forward(self, x):
                # needs your implementation
                # flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten()
                # f = nn.Flatten() # flatten to one-dimension
                # fx = f(x)
                fx = x.reshape(-1, 784)
                z = self.layers(fx)
                return z

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder_layers = nn.Sequential(
                    nn.Linear(
                        in_features=input_size, out_features=feature_size),
                    nn.Tanh()
                )

            def forward(self, z):
                # needs your implementation
                rec = self.decoder_layers(z)
                # reconstructed = torch.reshape(rec, (-1, 1, 28, 28))
                reconstructed = rec.reshape(rec.size(0), 1, 28, 28)
                return reconstructed

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    # Implement this function for the DAE model
    # def add_noise(self, x, noise_type):
    #     if noise_type=='Gaussian':
    #         gnoise = torch.randn_like(x)
    #         gnoise_x = x + gnoise
    #         return gnoise_x
    #     elif noise_type=='Dropout':
    #         d = nn.Dropout(p=0.5)
    #         dnoise = d(x)
    #         return dnoise

    # Implement this function for the VAE model
    # def reparameterise(self, mu, logvar):
    #     if self.training:
    #         # return reparametrized mu
    #     else:
    #         # return mu

    def forward(self, x):
        # This function should be modified for the DAE and VAE
        # noised_x = self.add_noise(x, self.noise_type)
        # new_x = torch.from_numpy(noised_x)
        x0 = self.encoder(x)
        x = self.decoder(x0)
        # for the VAE forward function should also return mu and logvar
        return x
```

Training code & Losses (no noise):

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share W

+ Code + Text

RAM Disk Editing

Training Code for Part 3

```

3m ① from autoencoder_starter import Autoencoder_Trainer

{x} LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

## uncomment for autoencoder:
autoer = Autoencoder(dim_latent_representation=30)
trainer = Autoencoder_Trainer(autoencoder_model=autoer,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

## uncomment for DAE with 2 noises:
# dae = DAE(dim_latent_representation=30, noise_type='Gaussian')
# dae = DAE(dim_latent_representation=30, noise_type='Dropout')
# trainer = Autoencoder_Trainer(autoencoder_model=dae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")

```

2m ② /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested cpuset_checked()
100% [██████████] 1875/1875 [00:15<00:00, 121.01it/s]==> Epoch: 1 Average loss: 0.5796

100% [██████████] 313/313 [00:02<00:00, 142.32it/s]==> Val set loss (reconstruction error) : 0.5256

{x} 100% [██████████] 1875/1875 [00:15<00:00, 122.44it/s]==> Epoch: 2 Average loss: 0.5163

100% [██████████] 313/313 [00:02<00:00, 142.18it/s]==> Val set loss (reconstruction error) : 0.5085

100% [██████████] 1875/1875 [00:15<00:00, 120.38it/s]==> Epoch: 3 Average loss: 0.5065

100% [██████████] 313/313 [00:02<00:00, 140.48it/s]==> Val set loss (reconstruction error) : 0.5034

100% [██████████] 1875/1875 [00:15<00:00, 123.37it/s]==> Epoch: 4 Average loss: 0.5039

100% [██████████] 313/313 [00:02<00:00, 145.22it/s]==> Val set loss (reconstruction error) : 0.5021

100% [██████████] 1875/1875 [00:15<00:00, 122.96it/s]==> Epoch: 5 Average loss: 0.5030

100% [██████████] 313/313 [00:02<00:00, 142.46it/s]==> Val set loss (reconstruction error) : 0.5017

100% [██████████] 1875/1875 [00:15<00:00, 122.72it/s]==> Epoch: 6 Average loss: 0.5026

100% [██████████] 313/313 [00:02<00:00, 139.06it/s]==> Val set loss (reconstruction error) : 0.5015

100% [██████████] 1875/1875 [00:15<00:00, 123.28it/s]==> Epoch: 7 Average loss: 0.5024

100% [██████████] 313/313 [00:02<00:00, 141.27it/s]==> Val set loss (reconstruction error) : 0.5014

100% [██████████] 1875/1875 [00:15<00:00, 122.42it/s]==> Epoch: 8 Average loss: 0.5023

100% [██████████] 313/313 [00:02<00:00, 144.56it/s]==> Val set loss (reconstruction error) : 0.5013

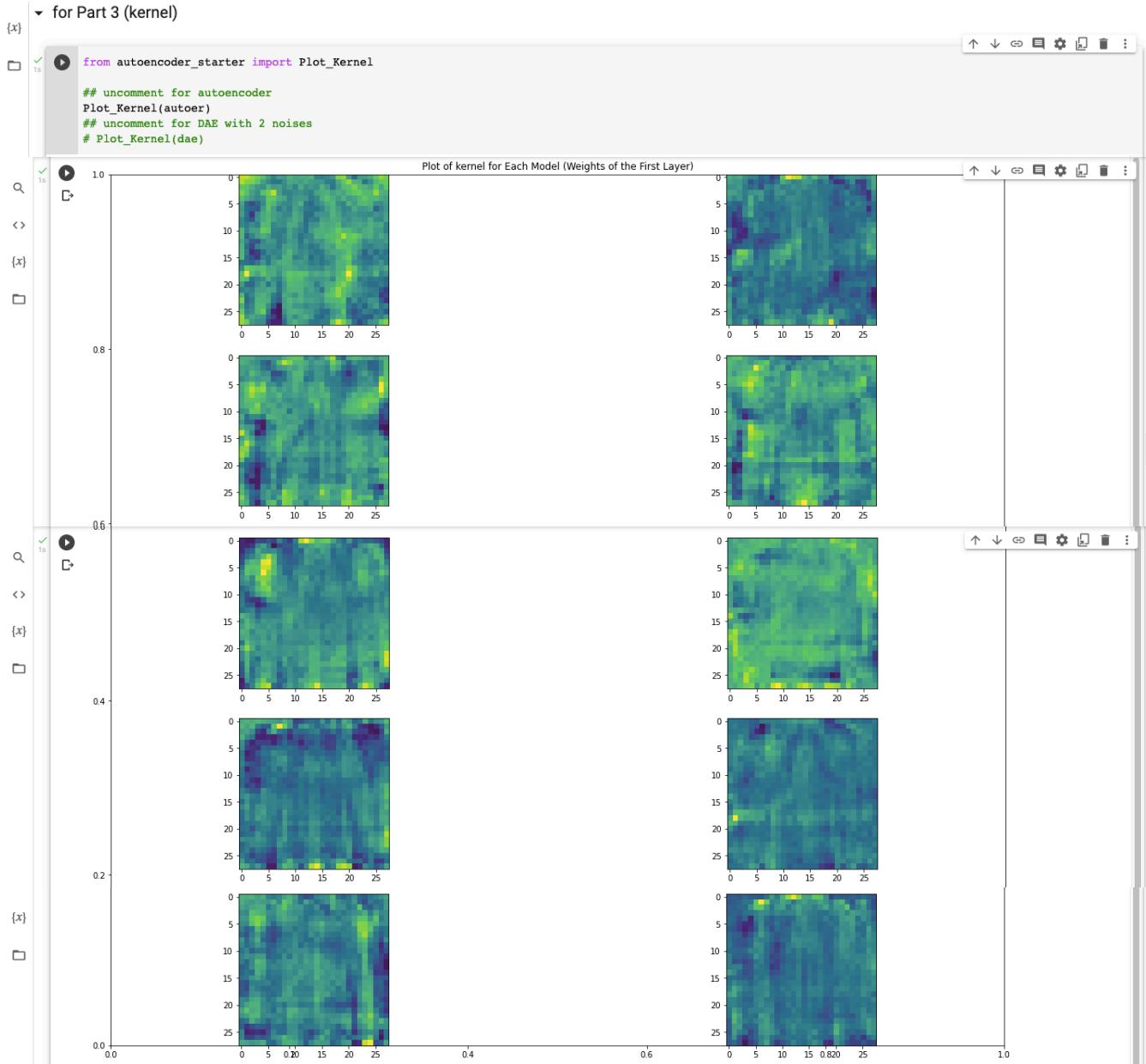
100% [██████████] 1875/1875 [00:15<00:00, 119.50it/s]==> Epoch: 9 Average loss: 0.5022

100% [██████████] 313/313 [00:02<00:00, 138.12it/s]==> Val set loss (reconstruction error) : 0.5013

100% [██████████] 1875/1875 [00:15<00:00, 121.35it/s]==> Epoch: 10 Average loss: 0.5021

100% [██████████] 313/313 [00:02<00:00, 139.10it/s]==> Val set loss (reconstruction error) : 0.5012

Kernel plots:



DAE with Dropout noise

```

  Chrome File Edit View History Bookmarks Profiles Tab Window Help MEM 74% 
  Happen x 在线翻译 x CMPT 7 x Unders x Pocket x Pytorch x Getting x pytorch x Guide to x Mail - w x Autoenc x + 
  colab.research.google.com/drive/1_-PsRRYZFUDm9Aa50ShCUXIV4Zx1xbtd#scrollTo=Hou54W2B0V1x 
  Apps Gengo translation... Pocket The Wall Street Jo... The Economist Google 艺术与文化 Synthetic Financia... Find Courses by T... Mac Keyboard Sy... 
  Mon Dec 6 13:41 
  Autoencoder_sample.ipynb 
  File Edit View Insert Runtime Tools Help All changes saved 
  Comment Share W 
  RAM Disk Editing 
  + Code + Text 
  Part 3 
  class DAE(nn.Module): 
      def __init__(self, dim_latent_representation=2, noise_type='Gaussian'): 
          self.noise_type = noise_type 
          super(DAE, self).__init__() 

          class Encoder(nn.Module): 
              def __init__(self, output_size=2): 
                  super(Encoder, self).__init__() 
                  # needs your implementation 
                  self.layers = nn.Sequential( 
                      nn.Linear(in_features=feature_size, out_features=output_size) 
                  ) 

              def forward(self, x): 
                  # needs your implementation 
                  # flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten() 
                  # f = nn.Flatten() # flatten to one-dimension 
                  # fx = f(x) 
                  fx = x.reshape(-1, 784) 
                  z = self.layers(fx) 
                  return z 

      def __init__(self, input_size=2): 
          super(Decoder, self).__init__() 
          # needs your implementation 
          self.decoder_layers = nn.Sequential( 
              nn.Linear( 
                  in_features=input_size, out_features=feature_size), 
              nn.Tanh() 
          ) 

          def forward(self, z): 
              # needs your implementation 
              rec = self.decoder_layers(z) 
              # reconstructed = torch.reshape(rec, (-1, 1, 28, 28)) 
              reconstructed = rec.reshape(rec.size(0), 1, 28, 28) 
              return reconstructed 

      encoder = Encoder(output_size=dim_latent_representation) 
      decoder = Decoder(input_size=dim_latent_representation) 

  # Implement this function for the DAE model 
  def add_noise(self, x, noise_type): 
      if noise_type == 'Gaussian': 
          gnoise = torch.randn_like(x) 
          gnoise_x = x + gnoise 
          return gnoise_x 
      elif noise_type == 'Dropout': 
          d = nn.Dropout(p=0.5) 
          dnoise = d(x) 
          return dnoise 

  # Implement this function for the VAE model 
  # def reparameterise(self, mu, logvar): 
  #     if self.training: 
  #         # return reparametrized mu 
  #     else: 
  #         return mu 

  def forward(self, x): 
      # This function should be modified for the DAE and VAE 
      noised_x = self.add_noise(x, self.noise_type) 
      new_x = torch.from_numpy(noised_x) 
      x0 = self.encoder(new_x) 
      x = self.decoder(x0) 
      # for the VAE forward function should also return mu and logvar 
      return x

```

Training code & Losses for Dropout noise:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share ⚙ W

+ Code + Text

Training Code for Part 3

```

from autoencoder_starter import Autoencoder_Trainer

LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

## uncomment for autoencoder:
# autoer = Autoencoder(dim_latent_representation=30)
# trainer = Autoencoder_Trainer(autoencoder_model=autoer,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

## uncomment for DAE with 2 noises:
# dae = DAE(dim_latent_representation=30, noise_type='Gaussian')
# dae = DAE(dim_latent_representation=30, noise_type='Dropout')
trainer = Autoencoder_Trainer(autoencoder_model=dae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
    except (KeyboardInterrupt, SystemExit):
        print("Manual Interruption")

```

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share ⚙ W

+ Code + Text

2m ✓ RAM Disk Editing

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes. Consider using num_workers=0 or pin_memory=True to avoid this warning.

100% [██████████] 1875/1875 [00:15<00:00, 120.28it/s]==> Epoch: 1 Average loss: 0.5982

100% [██████████] 313/313 [00:02<00:00, 142.67it/s]==> Val set loss (reconstruction error) : 0.5459

100% [██████████] 1875/1875 [00:15<00:00, 122.75it/s]==> Epoch: 2 Average loss: 0.5392

100% [██████████] 313/313 [00:02<00:00, 141.15it/s]==> Val set loss (reconstruction error) : 0.5326

100% [██████████] 1875/1875 [00:15<00:00, 122.48it/s]==> Epoch: 3 Average loss: 0.5301

100% [██████████] 313/313 [00:02<00:00, 144.89it/s]==> Val set loss (reconstruction error) : 0.5264

100% [██████████] 1875/1875 [00:15<00:00, 122.53it/s]==> Epoch: 4 Average loss: 0.5267

100% [██████████] 313/313 [00:02<00:00, 143.59it/s]==> Val set loss (reconstruction error) : 0.5252

100% [██████████] 1875/1875 [00:15<00:00, 124.95it/s]==> Epoch: 5 Average loss: 0.5259

100% [██████████] 313/313 [00:02<00:00, 144.58it/s]==> Val set loss (reconstruction error) : 0.5250

100% [██████████] 1875/1875 [00:15<00:00, 122.84it/s]==> Epoch: 6 Average loss: 0.5258

100% [██████████] 313/313 [00:02<00:00, 143.27it/s]==> Val set loss (reconstruction error) : 0.5250

100% [██████████] 1875/1875 [00:15<00:00, 123.41it/s]==> Epoch: 7 Average loss: 0.5256

100% [██████████] 313/313 [00:02<00:00, 138.33it/s]==> Val set loss (reconstruction error) : 0.5248

100% [██████████] 1875/1875 [00:15<00:00, 121.26it/s]==> Epoch: 8 Average loss: 0.5256

100% [██████████] 313/313 [00:02<00:00, 140.41it/s]==> Val set loss (reconstruction error) : 0.5247

100% [██████████] 1875/1875 [00:15<00:00, 122.22it/s]==> Epoch: 9 Average loss: 0.5255

100% [██████████] 313/313 [00:02<00:00, 141.98it/s]==> Val set loss (reconstruction error) : 0.5245

100% [██████████] 1875/1875 [00:15<00:00, 120.42it/s]==> Epoch: 10 Average loss: 0.5255

100% [██████████] 313/313 [00:02<00:00, 137.88it/s]==> Val set loss (reconstruction error) : 0.5247

Kernel plots:

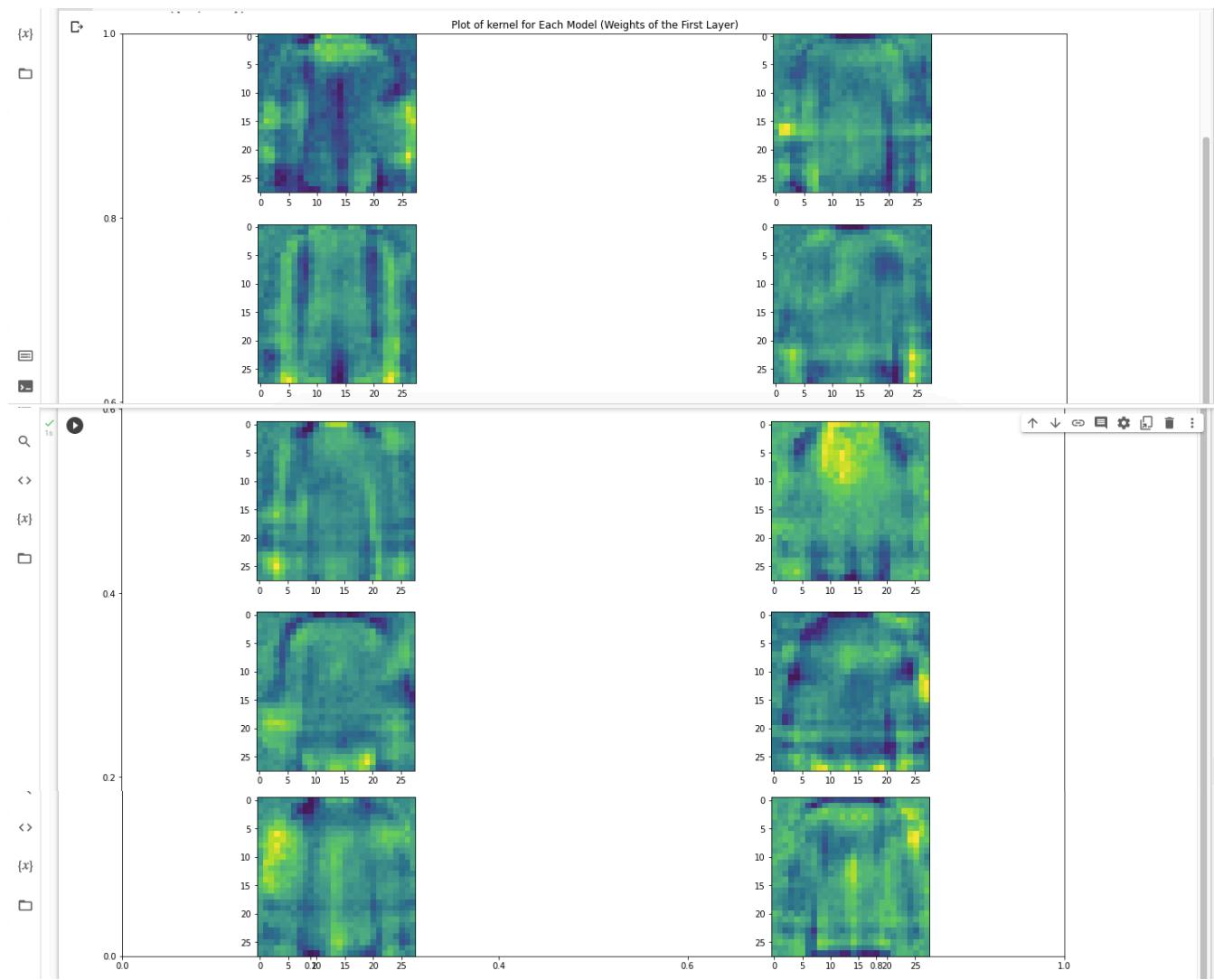
for Part 3 (kernel)

```

[32] from autoencoder_starter import Plot_Kernel

## uncomment for autoencoder
# Plot_Kernel(autoer)
## uncomment for DAE with 2 noises
Plot_Kernel(dae)

```



DAE with Gaussian noise

Training code & Losses for Gaussian noise:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share ⚙ W

+ Code + Text

RAM Disk Editing

Training Code for Part 3

```

from autoencoder_starter import Autoencoder_Trainer

LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

## uncomment for autoencoder:
# autoer = Autoencoder(dim_latent_representation=30)
# trainer = Autoencoder_Trainer(autoencoder_model=autoer,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

## uncomment for DAE with 2 noises:
dae = DAE(dim_latent_representation=30, noise_type='Gaussian')
# dae = DAE(dim_latent_representation=30, noise_type='Dropout')
trainer = Autoencoder_Trainer(autocencoder_model=dae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")

```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes. Consider using num_workers=0 or cpuset_checked.

100% [██████████] 1875/1875 [00:15<00:00, 119.85it/s]==> Epoch: 1 Average loss: 0.5989

100% [██████████] 313/313 [00:02<00:00, 140.76it/s]==> Val set loss (reconstruction error) : 0.5485

100% [██████████] 1875/1875 [00:15<00:00, 120.45it/s]==> Epoch: 2 Average loss: 0.5423

100% [██████████] 313/313 [00:02<00:00, 139.29it/s]==> Val set loss (reconstruction error) : 0.5358

100% [██████████] 1875/1875 [00:15<00:00, 118.94it/s]==> Epoch: 3 Average loss: 0.5341

100% [██████████] 313/313 [00:02<00:00, 136.61it/s]==> Val set loss (reconstruction error) : 0.5308

100% [██████████] 1875/1875 [00:15<00:00, 117.87it/s]==> Epoch: 4 Average loss: 0.5310

100% [██████████] 313/313 [00:02<00:00, 137.41it/s]==> Val set loss (reconstruction error) : 0.5295

100% [██████████] 1875/1875 [00:16<00:00, 117.06it/s]==> Epoch: 5 Average loss: 0.5303

100% [██████████] 313/313 [00:02<00:00, 141.34it/s]==> Val set loss (reconstruction error) : 0.5295

100% [██████████] 1875/1875 [00:15<00:00, 121.20it/s]==> Epoch: 6 Average loss: 0.5301

100% [██████████] 313/313 [00:02<00:00, 139.07it/s]==> Val set loss (reconstruction error) : 0.5297

100% [██████████] 1875/1875 [00:15<00:00, 120.91it/s]==> Epoch: 7 Average loss: 0.5301

100% [██████████] 313/313 [00:02<00:00, 137.79it/s]==> Val set loss (reconstruction error) : 0.5292

100% [██████████] 1875/1875 [00:15<00:00, 119.68it/s]==> Epoch: 8 Average loss: 0.5300

100% [██████████] 313/313 [00:02<00:00, 139.88it/s]==> Val set loss (reconstruction error) : 0.5295

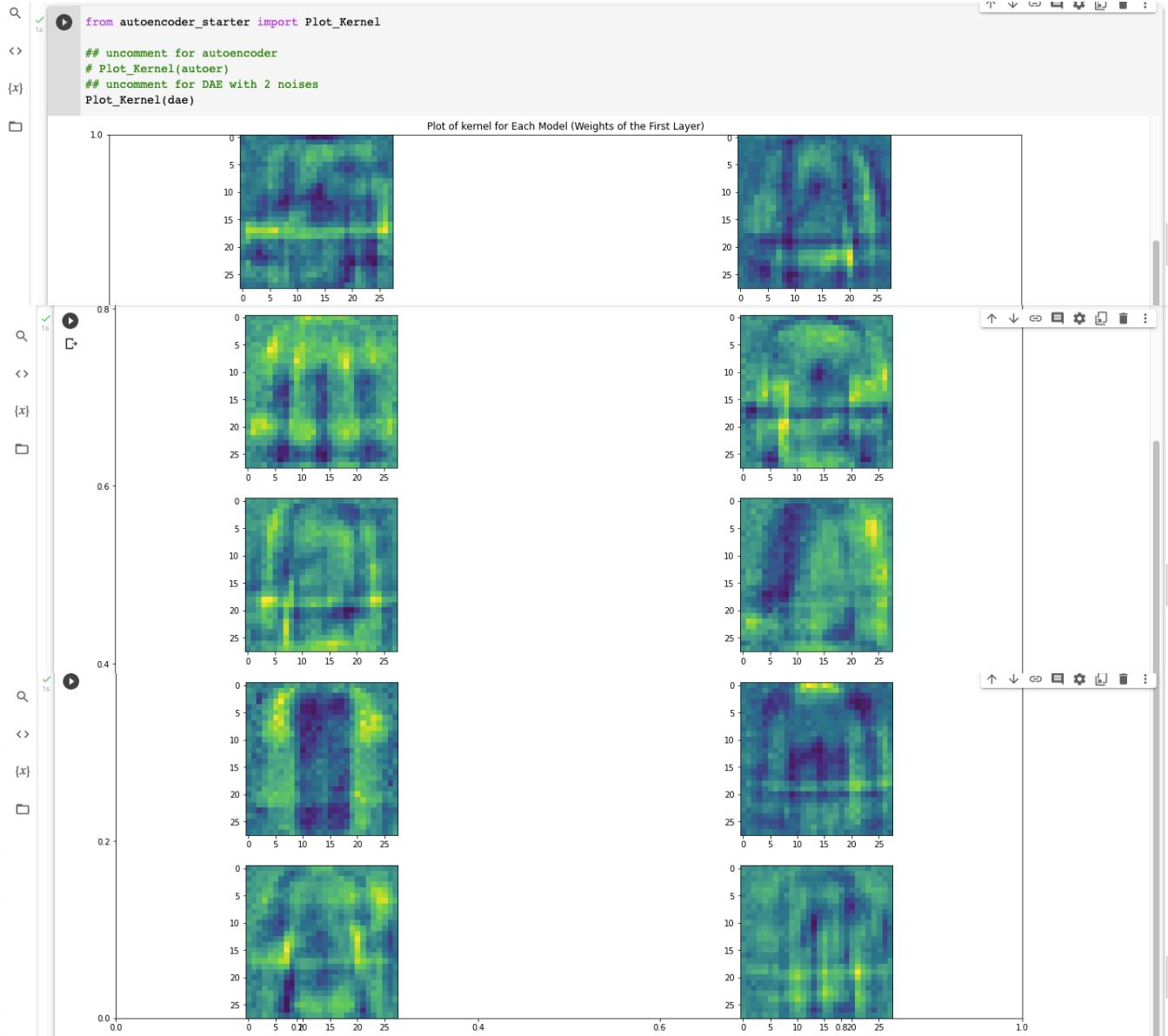
100% [██████████] 1875/1875 [00:15<00:00, 119.67it/s]==> Epoch: 9 Average loss: 0.5299

100% [██████████] 313/313 [00:02<00:00, 141.12it/s]==> Val set loss (reconstruction error) : 0.5293

100% [██████████] 1875/1875 [00:15<00:00, 119.83it/s]==> Epoch: 10 Average loss: 0.5298

100% [██████████] 313/313 [00:02<00:00, 139.17it/s]==> Val set loss (reconstruction error) : 0.5290

Kernel plots:



Describe how the kernels differ from each other and explain what this says about the denoising autoencoder.

Kernel of autoencoder with no noise removed is less clear and obvious than kernel of DAE. Kernel of DAE with Gaussian noise is a bit more obviously than kernel of DAE with Dropout noise. We can more easily define the shape of images in kernel of DAE.

This implies that removing noise from input enables the network to ignore less important features (“noise”) in images and learn more important features from the images. By removing noises, the risk of learning the identity function and learning from noises instead of extracting key features is reduced. Therefore, DAE learns and performs better than autoencoder without noises removed.

#4

Autoencoder

```

Autoencoder_sample.ipynb
File Edit View Insert Runtime Tools Help All changes saved

Part 4 (just autoencoder)

[11] class Autoencoder(nn.Module):
    def __init__(self, dim_latent_representation=2):
        super(Autoencoder, self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=2):
                super(Encoder, self).__init__()
                # needs your implementation
                self.layers = nn.Sequential(
                    nn.Linear(in_features=feature_size, out_features=output_size)
                )

            def forward(self, x):
                # needs your implementation
                # flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten()
                # f = nn.Flatten() # flatten to one-dimension
                # fx = f(x)
                fx = x.reshape(-1, 784)
                z = self.layers(fx)
                return z

        class Decoder(nn.Module):
            def __init__(self, input_size=2):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder_layers = nn.Sequential(
                    nn.Linear(
                        in_features=input_size, out_features=feature_size),
                    nn.Tanh()
                )

            def forward(self, z):
                # needs your implementation
                rec = self.decoder_layers(z)
                # reconstructed = torch.reshape(rec, (-1, 1, 28, 28))
                reconstructed = rec.reshape(rec.size(0), 1, 28, 28)
                return reconstructed

        self.encoder = Encoder(output_size=dim_latent_representation)
        self.decoder = Decoder(input_size=dim_latent_representation)

    # Implement this function for the DAE model
    # def add_noise(self, x, noise_type):
    #     if noise_type=='Gaussian':
    #         gnoise = torch.randn_like(x)
    #         gnoise_x = x + gnoise
    #         return gnoise_x
    #     elif noise_type=='Dropout':
    #         d = nn.Dropout(p=0.5)
    #         dnoise = d(x)
    #         return dnoise

    # Implement this function for the VAE model
    # def reparameterize(self, mu, logvar):
    #     if self.training:
    #         # return reparametrized mu
    #     else:
    #         return mu

    def forward(self, x):
        # This function should be modified for the DAE and VAE
        # noised_x = self.add_noise(x, self.noise_type)
        # new_x = torch.from_numpy(noised_x)
        x0 = self.encoder(x)
        x = self.decoder(x0)
        # for the VAE forward function should also return mu and logvar
        return x

```

Training code:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share W

+ Code + Text

Training code for Part 4

```

from VAE_starter import VAE_Trainer
from autoencoder_starter import Autoencoder_Trainer

{x}
LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

# for autoencoder
ae = Autoencoder(dim_latent_representation=30)
trainer = Autoencoder_Trainer(autoencoder_model=ae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)
# for VAE
# vae = VAE(dim_latent_representation=30)
# trainer = VAE_Trainer(autoencoder_model=vae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")

```

Losses:

```

[...]
  /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested
  cpuset_checked()
100%|██████████| 1875/1875 [00:14<00:00, 126.15it/s]==> Epoch: 1 Average loss: 0.5809
100%|██████████| 313/313 [00:02<00:00, 147.72it/s]==> Val set loss (reconstruction error) : 0.5258
100%|██████████| 1875/1875 [00:14<00:00, 125.57it/s]==> Epoch: 2 Average loss: 0.5162
100%|██████████| 313/313 [00:02<00:00, 146.09it/s]==> Val set loss (reconstruction error) : 0.5077
100%|██████████| 1875/1875 [00:14<00:00, 126.28it/s]==> Epoch: 3 Average loss: 0.5062
100%|██████████| 313/313 [00:02<00:00, 146.02it/s]==> Val set loss (reconstruction error) : 0.5032
100%|██████████| 1875/1875 [00:15<00:00, 124.92it/s]==> Epoch: 4 Average loss: 0.5037
100%|██████████| 313/313 [00:02<00:00, 146.09it/s]
==> Val set loss (reconstruction error) : 0.5019
100%|██████████| 1875/1875 [00:14<00:00, 125.80it/s]==> Epoch: 5 Average loss: 0.5029
100%|██████████| 313/313 [00:02<00:00, 146.90it/s]==> Val set loss (reconstruction error) : 0.5015
100%|██████████| 1875/1875 [00:15<00:00, 124.49it/s]==> Epoch: 6 Average loss: 0.5026
100%|██████████| 313/313 [00:02<00:00, 143.59it/s]==> Val set loss (reconstruction error) : 0.5014
100%|██████████| 1875/1875 [00:14<00:00, 125.77it/s]==> Epoch: 7 Average loss: 0.5024
100%|██████████| 313/313 [00:02<00:00, 146.99it/s]==> Val set loss (reconstruction error) : 0.5012
100%|██████████| 1875/1875 [00:15<00:00, 122.89it/s]==> Epoch: 8 Average loss: 0.5023
100%|██████████| 313/313 [00:02<00:00, 144.24it/s]==> Val set loss (reconstruction error) : 0.5011
100%|██████████| 1875/1875 [00:15<00:00, 124.18it/s]==> Epoch: 9 Average loss: 0.5022
100%|██████████| 313/313 [00:02<00:00, 146.76it/s]==> Val set loss (reconstruction error) : 0.5011
100%|██████████| 1875/1875 [00:15<00:00, 123.38it/s]==> Epoch: 10 Average loss: 0.5021
100%|██████████| 313/313 [00:02<00:00, 145.67it/s]==> Val set loss (reconstruction error) : 0.5010

```

latent space plot code:

```

for Part 4 (TSNE - autoencoder)

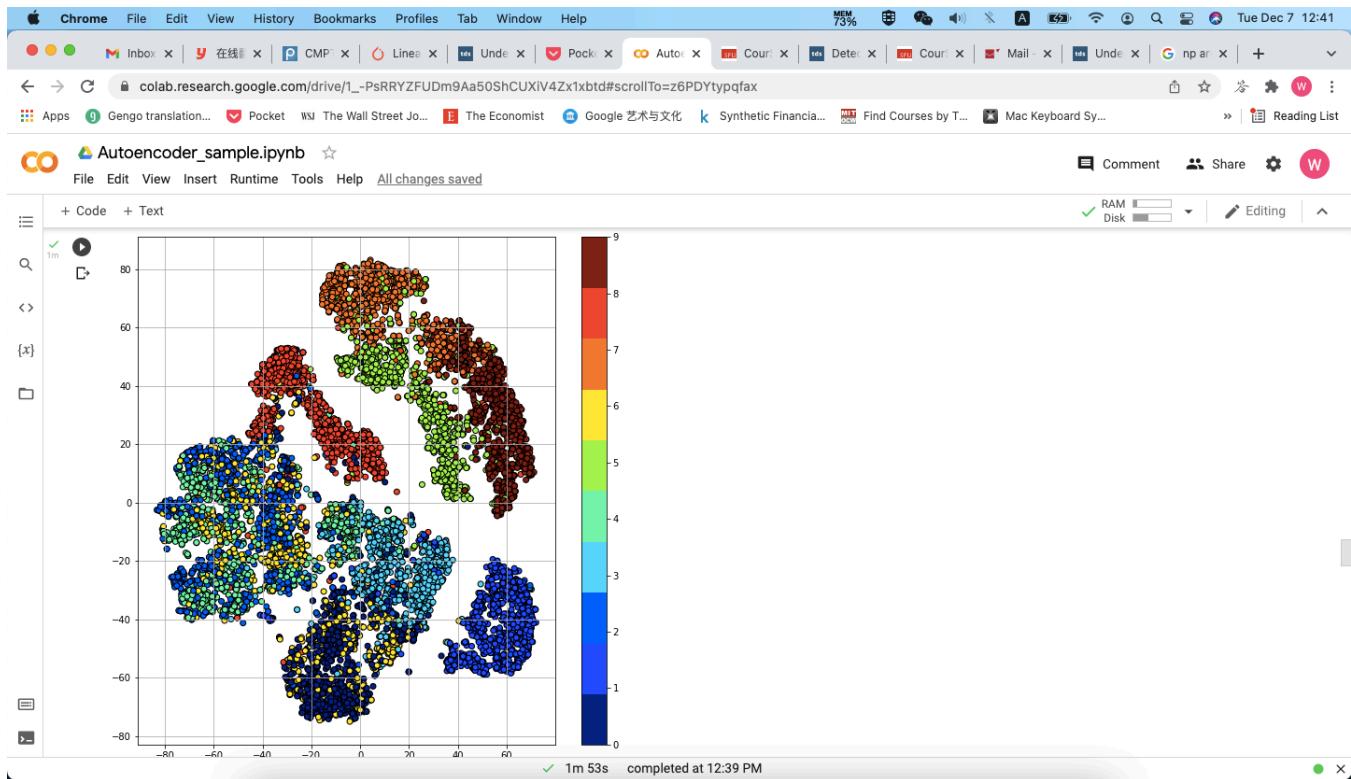
< > 1m with torch.no_grad():
    model = trainer.model
    model.eval()
    z=[];label=[]
    for x,y in trainer.val_loader:
        z_ = model.encoder(x.to(trainer.device))
        z += z_.cpu().tolist()
        label += y.cpu().tolist()
    z = np.asarray(z)
    print('this is z shape:')
    print(z.shape)
    label = np.asarray(label)

    from autoencoder_starter import scatter_plot
    from sklearn.manifold import TSNE

    tsne = TSNE(n_components=2)
    tsne_results = tsne.fit_transform(z)
    scatter_plot(latent_representations=tsne_results,labels=label)

```

latent space plot:



VAE

Here, I use KLD loss term with mean, which is $-0.5 * \text{torch.mean}(1 + \logvar - \mu.\text{pow}(2) - \logvar.\text{exp}())$, since this definition makes more sense in the learning result than the one with sum.

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Tue Dec 7 1:55

colab.research.google.com/drive/1_-PsRRYZFUDm9Aa50ShCUXIV4Zx1xbtd#scrollTo=TbHqrRH2FBDr

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

RAM Disk Editing

Part 4

```
[5] class VAE(nn.Module):
{x}     def __init__(self,dim_latent_representation=2):
        # self.noise_type = noise_type
        super(VAE,self).__init__()

        class Encoder(nn.Module):
            def __init__(self, output_size=dim_latent_representation):
                super(Encoder, self).__init__()
                # needs your implementation
                self.layers = nn.Sequential(
                    nn.Linear(in_features=28*28, out_features=dim_latent_representation)
                )
                self.fc1 = nn.Linear(dim_latent_representation, dim_latent_representation)
                self.fc2 = nn.Linear(dim_latent_representation, dim_latent_representation)

            def forward(self, x):
                # needs your implementation
                flattened_x = torch.flatten(x, start_dim=1) # same as nn.Flatten()
                # f = nn.Flatten() # flatten to one-dimension
                # fx = f(x)
                # fx = x.reshape(-1, 784)
                x_hat = self.layers(flattened_x)
                mu = self.fc1(x_hat)
                logvar = self.fc2(x_hat)
                return mu, logvar

        class Decoder(nn.Module):
            def __init__(self, input_size=dim_latent_representation):
                super(Decoder, self).__init__()
                # needs your implementation
                self.decoder_layers = nn.Sequential(
                    nn.Linear(
                        in_features=dim_latent_representation, out_features=feature_size),
                    nn.Tanh()
                )

            def forward(self, z):
                # needs your implementation
                rec = self.decoder_layers(z)
                reconstructed = rec.view(-1,1,28,28)
                # reconstructed = rec.reshape(rec.size(0), 1, 28, 28)
                return reconstructed
```

```

    self.encoder = Encoder(output_size=dim_latent_representation)
    self.decoder = Decoder(input_size=dim_latent_representation)

    # # Implement this function for the DAE model
    # def add_noise(self, x, noise_type):
    #     if noise_type=='Gaussian':
    #         gnoise = torch.randn_like(x)
    #         gnoise_x = x + gnoise
    #         return gnoise_x
    #     elif noise_type=='Dropout':
    #         d = nn.Dropout(p=0.5)
    #         dnoise = d(x)
    #         return dnoise

    # Implement this function for the VAE model
    def reparameterise(self, mu, logvar):
        if self.training:
            std = torch.exp(logvar/2)
            eps = torch.randn_like(std)
            reparametrized_mu = mu + eps * std
            return reparametrized_mu
        else:
            return mu

[5]   def forward(self, x):
        # This function should be modified for the DAE and VAE
        # noised_x = self.add_noise(x, self.noise_type)
        # new_x = torch.from_numpy(noised_x)
        # x0 = self.encoder(new_x)
        # x1 = self.decoder(x0)
        # for the VAE forward function should also return mu and logvar
        mu, logvar = self.encoder(x)
        # logvar = self.encoder(x)
        z = self.reparameterise(mu, logvar)
        reconstruction = self.decoder(z)
        return reconstruction, mu, logvar

```

Training Code:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Training code for Part 4

```

from VAE_starter import VAE_Trainer
from autoencoder_starter import Autoencoder_Trainer

LEARNING_RATE = 1e-3
EPOCH_NUMBER= 10 # the number of epochs and learning rate can be tuned.

# for autoencoder
ae = Autoencoder(dim_latent_representation=30)
trainer = Autoencoder_Trainer(autoencoder_model=ae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)
# for VAE
# vae = VAE(dim_latent_representation=30)
# trainer = VAE_Trainer(autoencoder_model=vae,learning_rate=LEARNING_RATE,path_prefix=path_prefix)

try:
    for epoch in range(1, EPOCH_NUMBER + 1):
        trainer.train(epoch)
        trainer.validate(epoch)
except (KeyboardInterrupt, SystemExit):
    print("Manual Interruption")

```

Losses:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share W

RAM Disk Editing

```
3m 100% /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested configuration is to use num_workers=0 and pin_memory=True. See https://pytorch.org/docs/stable/_modules/torch/utils/data/dataloader.html#DataLoader for more information.
  cpuset_checked)
  100%|██████████| 1875/1875 [00:17<00:00, 105.65it/s]==> Epoch: 1 Average loss: 0.9517
  100%|██████████| 313/313 [00:02<00:00, 131.32it/s]==> Val set loss (reconstruction error) : 0.7728
{x} 100%|██████████| 1875/1875 [00:17<00:00, 107.13it/s]==> Epoch: 2 Average loss: 0.8291
  100%|██████████| 313/313 [00:02<00:00, 129.98it/s]==> Val set loss (reconstruction error) : 0.7372
  100%|██████████| 1875/1875 [00:17<00:00, 107.48it/s]==> Epoch: 3 Average loss: 0.7959
  100%|██████████| 313/313 [00:02<00:00, 133.58it/s]==> Val set loss (reconstruction error) : 0.7368
  100%|██████████| 1875/1875 [00:17<00:00, 106.96it/s]==> Epoch: 4 Average loss: 0.7896
  100%|██████████| 313/313 [00:02<00:00, 133.18it/s]==> Val set loss (reconstruction error) : 0.7387
  100%|██████████| 1875/1875 [00:17<00:00, 107.78it/s]==> Epoch: 5 Average loss: 0.7844
  100%|██████████| 313/313 [00:02<00:00, 133.55it/s]==> Val set loss (reconstruction error) : 0.7361
  100%|██████████| 1875/1875 [00:17<00:00, 106.76it/s]==> Epoch: 6 Average loss: 0.7809
  100%|██████████| 313/313 [00:02<00:00, 131.55it/s]==> Val set loss (reconstruction error) : 0.7325
  100%|██████████| 1875/1875 [00:17<00:00, 106.86it/s]==> Epoch: 7 Average loss: 0.7800
  100%|██████████| 313/313 [00:02<00:00, 129.79it/s]==> Val set loss (reconstruction error) : 0.7357
  100%|██████████| 1875/1875 [00:17<00:00, 106.40it/s]==> Epoch: 8 Average loss: 0.7800
  100%|██████████| 313/313 [00:02<00:00, 128.90it/s]==> Val set loss (reconstruction error) : 0.7353
  100%|██████████| 1875/1875 [00:17<00:00, 105.60it/s]==> Epoch: 9 Average loss: 0.7792
{x} 100%|██████████| 313/313 [00:02<00:00, 132.76it/s]==> Val set loss (reconstruction error) : 0.7321
  100%|██████████| 1875/1875 [00:17<00:00, 105.82it/s]==> Epoch: 10 Average loss: 0.7791
  100%|██████████| 313/313 [00:02<00:00, 130.97it/s]==> Val set loss (reconstruction error) : 0.7326
```

latent space plot code:

Autoencoder_sample.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share W

RAM Disk Editing

```
for Part 4 (TSNE)
```

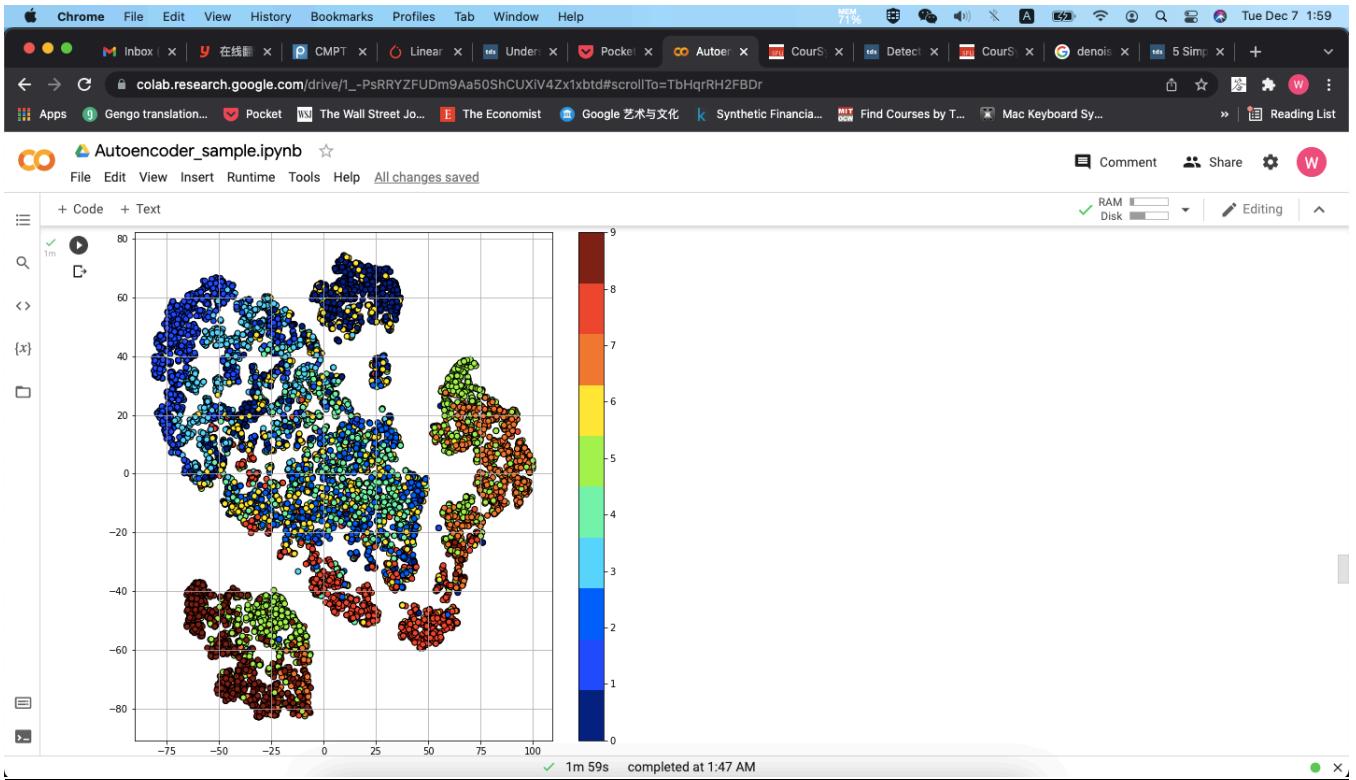
```
with torch.no_grad():
    model = trainer.model
    model.eval()
    z=[];label=[]
    for x,y in trainer.val_loader:

        mu, logvar = model.encoder(x.to(trainer.device))
        z_ = model.reparameterise(mu,logvar)
        z += z_.cpu().tolist()
        # z += z_.cpu().tolist()
        label += y.cpu().tolist()
    z = np.asarray(z)
    label = np.asarray(label)

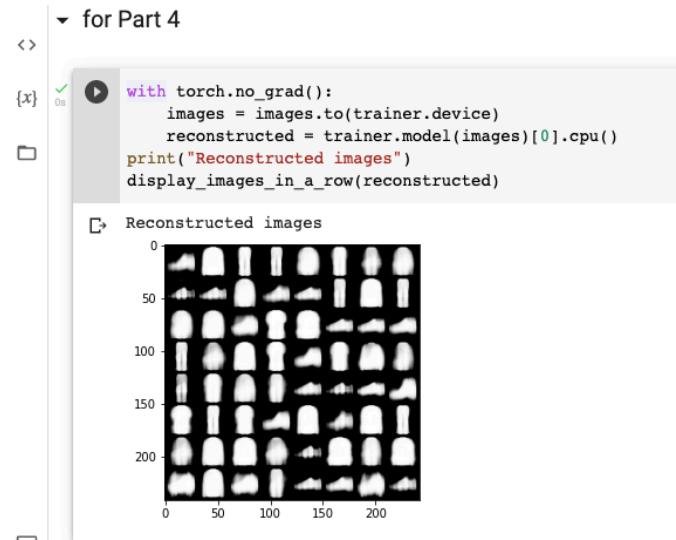
    from VAE_starter import scatter_plot
    from sklearn.manifold import TSNE

    tsne = TSNE(n_components=2)
    tsne_results = tsne.fit_transform(z)
    scatter_plot(latent_representations=tsne_results,labels=label)
```

latent space plot:



Reconstructed image:



Describe how the latent features are differ from each other for two models and explain what this says about the variational autoencoders.

Points with different colors gathers more compactly and the between-groups distance is larger in autoencoder than in VAE. Although it is obvious that points with same colors gather in VAE's plot and we can clearly define different color groups of points, their in-group distance is larger than latent

features in autoencoder does. It seems that autoencoder with 30-dimension latent space performs better in capturing the images than its counterpart VAE does. Autoencoder here may be overfitting.

First, VAE modeling with Gaussians allows each dimension in the representation to push themselves as farther as possible from the other factors. Therefore, not so compactly gather as autoencoder does. It also implies that VAE introduces regularization during training process for counterpart autoencoder and avoid overfitting to ensure that the latent space has good properties that enable generative purpose.