Anomaly Detection with Robust Deep Autoencoders

Group 10

Group Member:

Ying Peng

Ye Leng

Junjie Chu

UPPSALA
UNIVERSITET

The project content can be viewed in Google colab.

Agenda

1. Introduction of Theory
   1.1. Background (Autoencoder and RPCA)
   1.2. Introduction to RDAE
2. Code Demonstration
   2.1. Structure of the Program
   2.2. Deep autoencoder
   2.3. Variational autoencoder
   2.4. Robust Deep autoencoder
   2.5. Other Methods
3. Experiments
   3.1. Denoising
   3.2. Outlier Detection

UPPSALA
UNIVERSITET

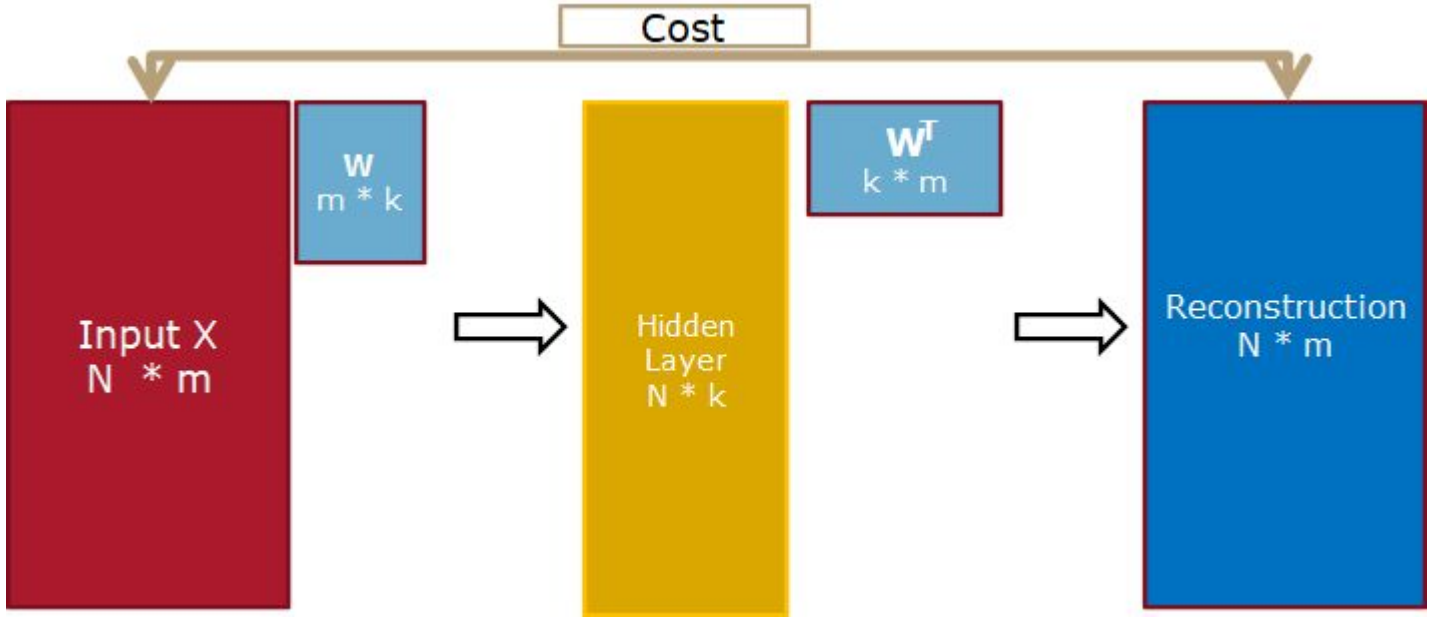
1. Introduction of Theory

- Background (Autoencoder and RPCA)
- Introduction to RDAE

1.1 Background (1/2)

Introduction to theory-Background-Autoencoder

- Autoencoders learn the input data themselves.
- Autoencoders become non-trivial by having the hidden layers are low-dimensional and non-linear.



Cost function:

$$\min_{D,E} \|X - D(E(X))\|$$

Where:

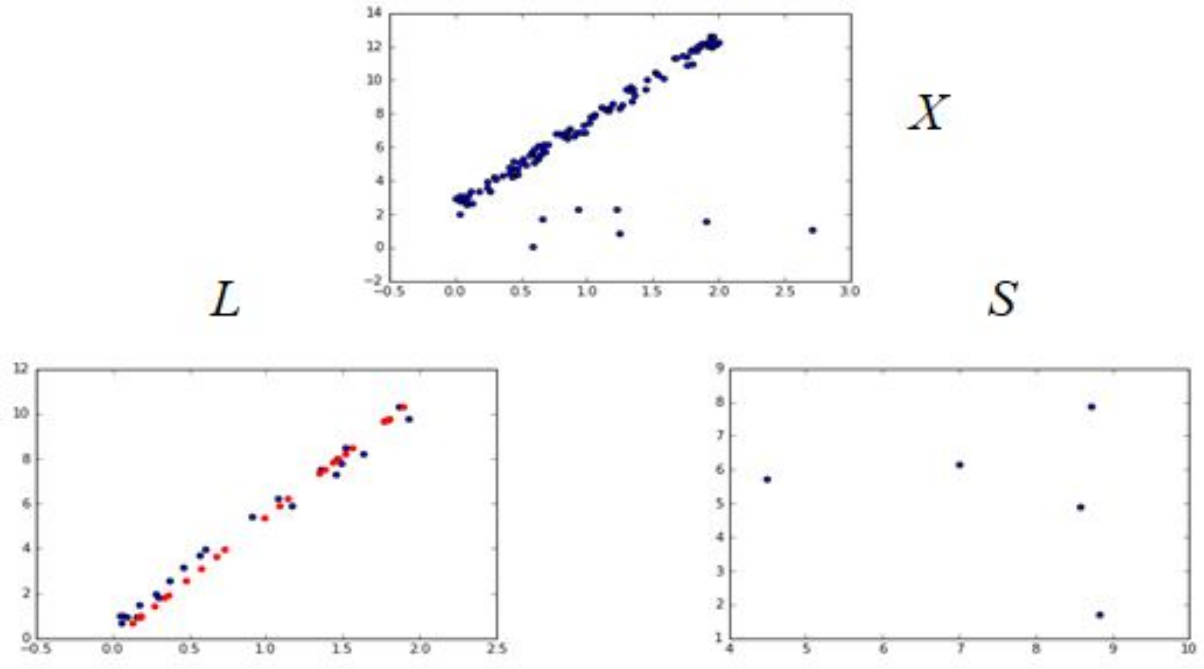
$$D_\theta(x) = D_{W,b}(x) = \text{logit}(W^T x + b_D)$$

$$E_\theta(x) = E_{W,b}(x) = \text{logit}(Wx + b_E)$$

1.1 Background (2/2)

Introduction to theory-Background-RPCA

- Robust Principal Component Analysis assumes that an image could be decomposed into two parts.
  - L contains structured data, thus it is a low-rank matrix. The lower the rank is, the better the components we find (if the number of components is constant).
  - S contains noise, which is unstructured. RPCA assumes that noise could be represented by a sparse matrix.



Finding low-rank matrix L and sparse matrix S can be transformed into an optimization problem of finding the minimum of the sum of the nuclear norm of L and the first norm of S:

$$\min_{L,S} ||L||_* + \lambda ||S||_1$$

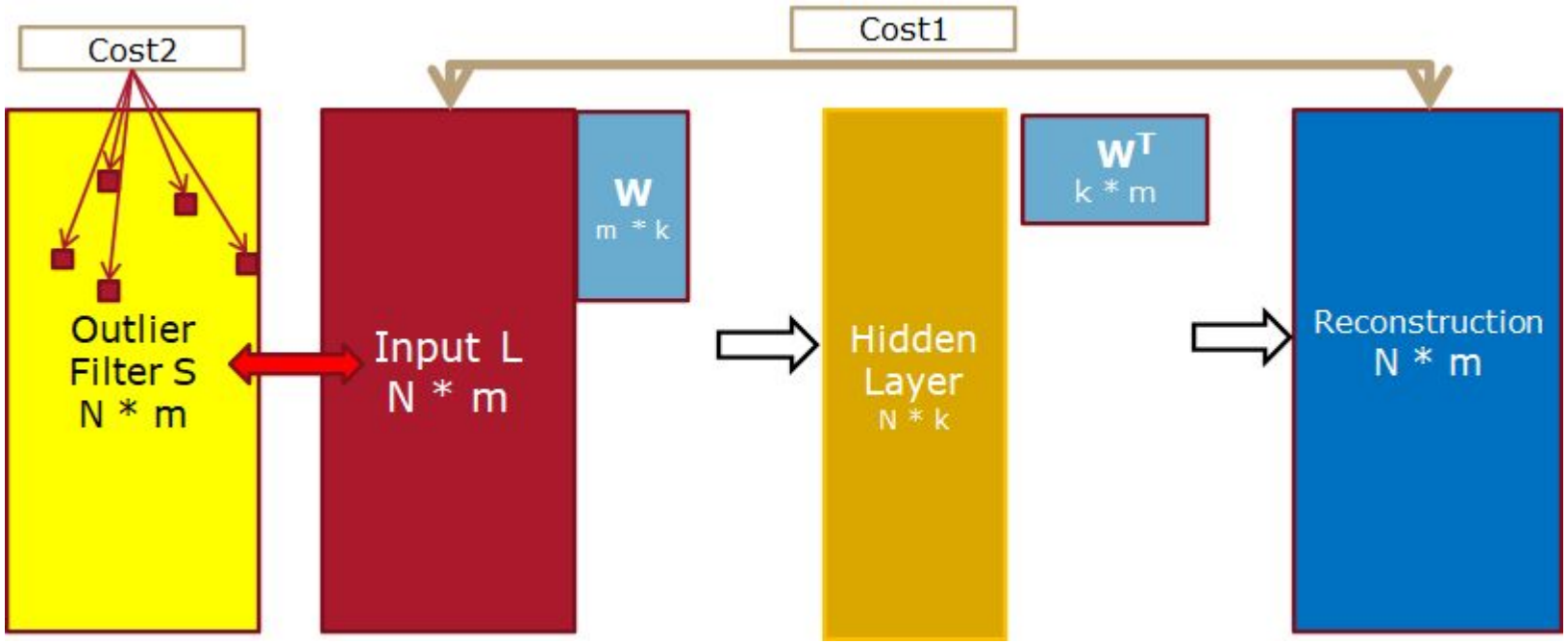
$$\text{s.t. } X - L - S = 0$$

UPPSALA UNIVERSITET

1.2.1 RDAE for denoising (1/1)

Introduction to theory-Introduction to RDAE-RDAE for denoising

- RDAE can be viewed as replacing the nuclear norm with a non-linear autoencoder.
  - The lowest-rank matrix L (i.e. no noise in L) has the minimum of nuclear norm.
  - The lowest-rank matrix L (i.e. no noise in L) could be reconstructed well via autoencoder.
  - Replace the nuclear norm with a non-linear autoencoder.



RPCA:

$$\min_{L,S} \boxed{||L||_*} + \lambda ||S||_1$$
$$\text{s.t. } X - L - S = 0$$

RDAE for denoising:

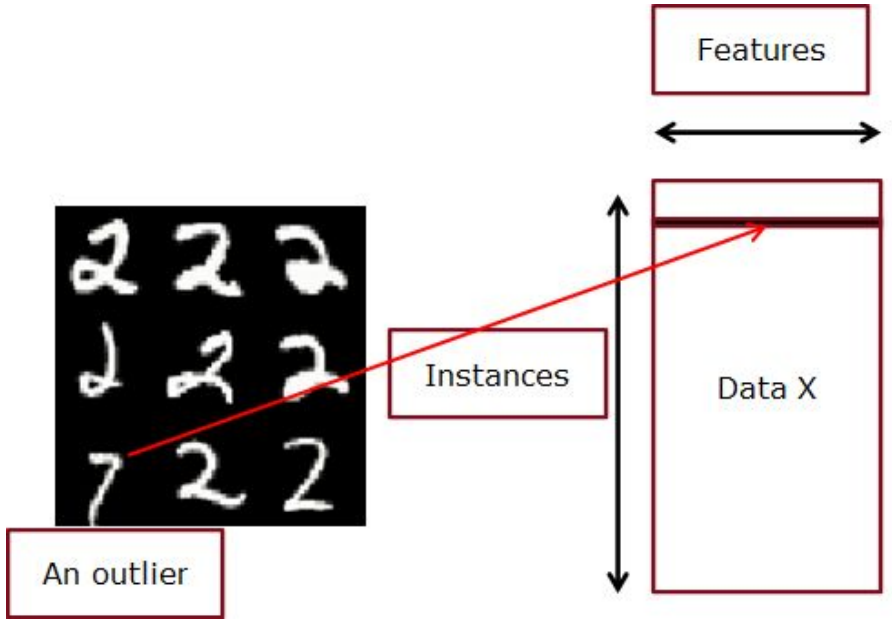
$$\min_{\theta} \boxed{\|L_D - D_\theta(E_\theta(L_D))\|_2} + \lambda \|S\|_1$$
$$\text{s.t. } X - L_D - S = 0.$$

UPPSALA UNIVERSITET

1.2.2 RDAE for outlier detection (1/2)

Introduction to theory-Introduction to RDAE-RDAE for outlier detection

- Noise is unstructured and thus we use l1 norm.
- Outliers are structured that an outlier is one row in a data matrix and we want to group elements over a row.
- Thus l1 norm is replaced by l2,1 norm.



RPCA:

$$\min_{L,S} ||L||_* + \lambda||S||_1$$
$$\text{s.t. } X - L - S = 0$$

RDAE for denoising:

$$\min_{\theta} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_1$$
$$\text{s.t. } X - L_D - S = 0.$$

$$||X||_{2,1} = \sum_{j=1}^{n}||x_j||_2 = \sum_{j=1}^{n}(\sum_{i=1}^{m}|x_{ij}|^2)^{1/2}$$

RDAE for outlier detection:

$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S^T\|_{2,1}$$
$$\text{s.t. } X - L_D - S = 0$$

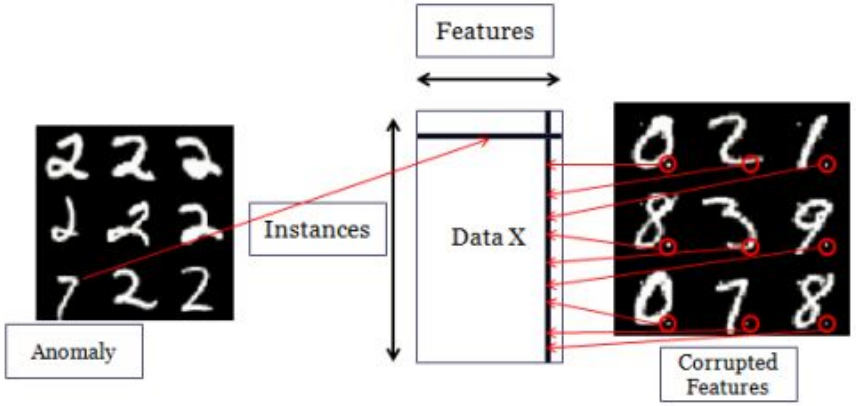
UPPSALA UNIVERSITET

1.2.2 RDAE for outlier detection (2/3)

Introduction to theory-Introduction to RDAE-RDAE for outlier detection

Detail of l2,1 norm

- S = X - L
- The 2-norm of each column(feature) in S is computed and then compared with Lambda.
  - If the norm is larger than Lambda, the values of the column will be kept and processed.
  - If the norm is smaller than Lambda, the values of the column will be set to 0.
- The 2-norm of each row(instance) in S is computed and then compared with Lambda.
  - Equal to each column in S^T
  - Operations are the same of above.
- The non-zero columns/rows in S are considered to be the outliers.



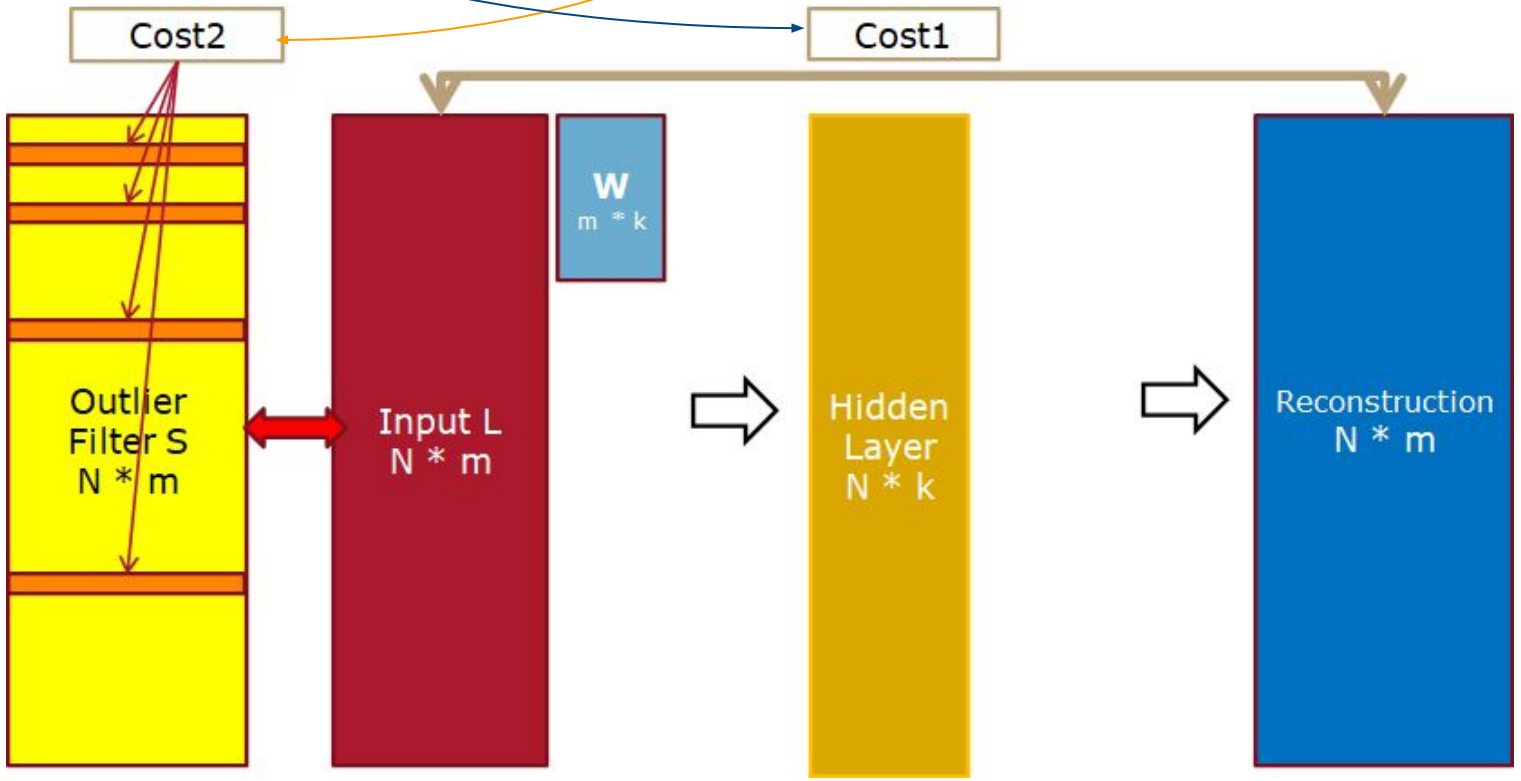
UPPSALA UNIVERSITET

1.2.2 RDAE for outlier detection (3/3)

Introduction to theory-Introduction to RDAE-RDAE for outlier detection

$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S^T\|_{2,1}$$

$$\text{s.t. } X - L_D - S = 0$$



1.2.3 Training and more details (1/2)

Introduction to theory-Introduction to RDAE-Training and more details

Training:

- Borrow the idea of ADMM that training each part of an complex object separately with other parts fixed.
  - Training the autoencoder with S fixed.
  - Shrink S with the autoencoder fixed.

$$\min_{\theta} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_1$$

$$\text{s.t. } X - L_D - S = 0.$$

$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S^T\|_{2,1}$$

$$\text{s.t. } X - L_D - S = 0$$

1.2.3 Training and more details (2/2)

Introduction to theory-Introduction to RDAE-Training and more details

More details:

- Minimize L with S fixed
  - Just a call to a standard autoencoder
- Project onto constraint
- Minimize S with L fixed
  - Proximal gradient problem

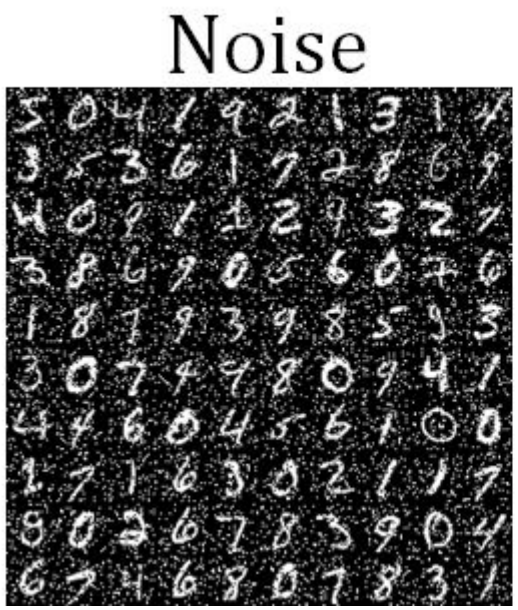
$$\min_{\theta} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_1 \quad \text{s.t. } X - L_D - S = 0.$$

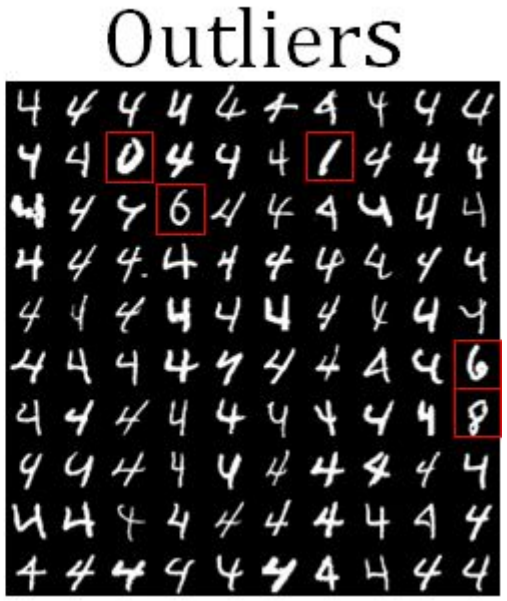
$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S^T\|_{2,1} \quad \text{s.t. } X - L_D - S = 0$$

- Project onto constraint
- Key points
  - Theoretical justification (the author still worked in progress)

1.2.4 Advantages (1/1)

Introduction to theory-Introduction to RDAE-Advantages





- Important point 1: Both problems can be handled in the ***same framework***
- Important point 2: Do ***not need any noise/outlier free training samples***

2. Code Demostration

- Structure of the Program
- Deep autoencoder
- Variational autoencoder
- Robust Deep autoencoder
- Other Methods

2.1 Structure of the Program

Language:
- Python

Tools:
- Google Colab

Models:
- Deep autoencoder
- Variational autoencoder
- Robust Deep autoencoder

Other methods:
- shrink
- add_noise
- compute_f1_score
- Imshow(MNIST tutorial)

2.2 Deep autoencoder

Class DAE(nn)

__init__()
encoder()
decoder()
forward()
cost_function()
fit()

```python
def __init__(self):
  super(DAE,self).__init__()

# encoder layers
  self.enc1 = nn.Linear(784, 625)
  self.enc2 = nn.Linear(625, 400)
  self.enc3 = nn.Linear(400, 225)
  self.enc4 = nn.Linear(225, 100)

  # decoder layers
  self.dec1 = nn.Linear(100, 255)
  self.dec2 = nn.Linear(255, 400)
  self.dec3 = nn.Linear(400, 625)
  self.dec4 = nn.Linear(625, 784)

  self.criterion = nn.MSELoss()
```

```python
def fit(self, X, optimizer):
  # move to the device
  X = X.to(device)

  # reset the gradient information
  optimizer.zero_grad()

  # compute loss
  loss = self.cost_function(X)

  # perform backpropagation
  loss.backward()

  # optimize the weight
  optimizer.step()

  return loss
```

```python
def encoder(self, x):
  x = F.relu(self.enc1(x))
  x = F.relu(self.enc2(x))
  x = F.relu(self.enc3(x))
  x = F.relu(self.enc4(x))

  return x
```

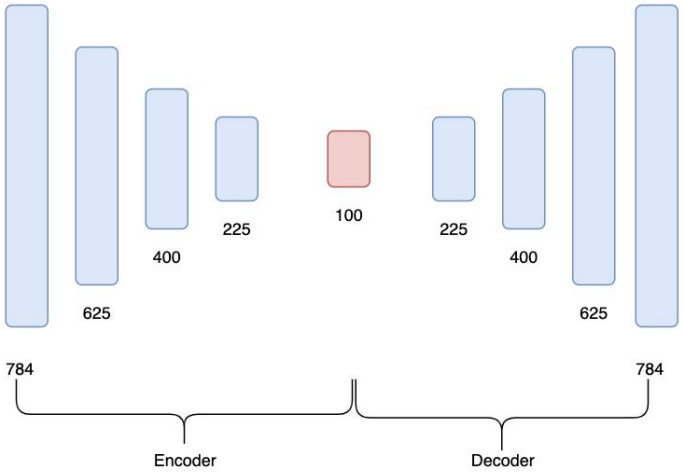
```python
def decoder(self, x):
  x = F.relu(self.dec1(x))
  x = F.relu(self.dec2(x))
  x = F.relu(self.dec3(x))
  x = F.relu(self.dec4(x))

  return x
```

```python
def forward(self, x):
  x = F.relu(self.enc1(x))
  x = F.relu(self.enc2(x))
  x = F.relu(self.enc3(x))
  x = F.relu(self.enc4(x))
  x = F.relu(self.dec1(x))
  x = F.relu(self.dec2(x))
  x = F.relu(self.dec3(x))
  x = F.relu(self.dec4(x))

  return x
```

```python
def cost_function(self,X):
  X = X.view(-1, 28*28)
  X = X.to(device)
  X_recon = self.forward(X)
  loss = self.criterion(X, X_recon)
  return loss
```



UPPSALA
UNIVERSITET

2.3 Variational autoencoder

Class VAE(nn)
__init__() *encoder()* *reparameterize()* *decoder()* *forward()* *cost_function()* *fit()*


```python
def __init__(self, image_size=784, h_dim=400, z_dim=20):
    super(VAE, self).__init__()
    self.fc1 = nn.Linear(image_size, h_dim)
    self.fc2 = nn.Linear(h_dim, z_dim)
    self.fc3 = nn.Linear(h_dim, z_dim)
    self.fc4 = nn.Linear(z_dim, h_dim)
    self.fc5 = nn.Linear(h_dim, image_size)

    self.criterion = nn.MSELoss()
```

```python
def fit(self, X, optimizer):
  # move to the device
  X = X.to(device)

  # reset the gradient information
  optimizer.zero_grad()

  # compute loss
  loss = self.cost_function(X)

  # perform backpropagation
  loss.backward()

  # optimize the weight
  optimizer.step()

  return loss
```

```python
# encoder
def encode(self, x):
    h = F.relu(self.fc1(x))
    return self.fc2(h), self.fc3(h)


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

# decoder
def decode(self, z):
    h = F.relu(self.fc4(z))
    return F.sigmoid(self.fc5(h))


def forward(self, x):
    mu, log_var = self.encode(x)
    z = self.reparameterize(mu, log_var)
    x_reconst = self.decode(z)
    return x_reconst

def cost_function(self,X):
  X = X.view(-1, 28*28)
  X = X.to(device)
  X_recon = self.forward(X)
  loss = self.criterion(X, X_recon)
  return loss
```

UPPSALA UNIVERSITET

2.4 Robust Deep autoencoder

Class RDAE(object)
__init__() *fit()*

```
def __init__(self, lambda_=1.0, error=1.0e-7):
  super(RDAE,self).__init__()
  self.lambda_ = lambda_
  self.error = error
  self.DAE = DAE.DAE()
```

```
def fit(self, X, iter=20, epoch=10):
  self.DAE.to(device)

  optimizer = optim.Adam(self.DAE.parameters(), lr=LEARNING_RATE)
  error = self.error
  lambda_ = self.lambda_

  L = torch.zeros(X.shape)
  S = torch.zeros(X.shape)

  LS0 = X

  XFnorm = torch.norm(X, 'fro')
  mu = (len(X))/(4.0 * torch.norm(X, 1))

  # move to the device
  X = X.to(device)
  L = L.to(device)
  S = S.to(device)
  LS0 = LS0.to(device)

  for i in range(iter):
    # updata L
    L = X - S

    # generate train data
    train_datas = torch.utils.data.DataLoader(L, batch_size = BATCH_SIZE, shuffle=False)

    train(train_datas, self.DAE, optimizer, epoch)

    with torch.no_grad():
      # get optimized L
      L = self.DAE(L.float())

      # S = shrink_anomly(lambda_/mu, (X-L))
      S = SHR.shrink(lambda_/mu, (X-L))

    # break criterion 1: the L and S are close enough to X
      c1 = torch.norm(X-L-S, 'fro') / XFnorm

    # break criterion 2: there is no changes for L and S
      c2 = torch.min(mu, torch.sqrt(mu)) * torch.norm(LS0 - L - S) / XFnorm

      if c1 < error and c2 < error:
        print("early break")
        break
    LS0 = L + S
  return L, S
```

Input: $X \in R^{N \times n}$

Initialize $L_D \in R^{N \times n}, S \in R^{N \times n}$ to be zero matrices, $LS = X$, and an autoencoder $D(E(\cdot))$ with randomly initialized parameters.

while(True):

1. Remove S from X, using the remainder to train the autoencoder. (In the first iteration, since S is the zero matrix, the autoencoder $D(E(\cdot))$ is given the input X):

$L_D = X - S$

2. Minimize the first term $||L_D - D(E(L_D))||_2$ using back-propagation.

3. Set L_D to be the reconstruction from the trained autoencoder:

$L_D = D(E(L_D))$

4. Set S to be the difference between X and L_D:

$S = X - L_D$

5. Optimize S using a proximal operator:

$S = prox_{\lambda, l_{2,1}}(S)$

or:

$S = prox_{\lambda, l_1}(S)$

6.1 Check the convergence condition that L_D and S are close to the input X thereby satisfying the constraint:

$c_1 = ||X - L_D - S||_2 \,/\, ||X||_2$

6.2 Check the convergence condition that L_D and S have converged to a fixed point:

$c_2 = ||LS - L_D - S||_2 \,/\, ||X||_2$

if $c_1 < \varepsilon$ or $c_2 < \varepsilon$:

break

7. Update LS for convergence checking in the next iteration:

$LS = L_D + S$

Return L_D and S

UNIVERSITET

2.5 Other Methods

Methods
shrink1
shrink2,1
add_noise
train
compute_f1_score

```python
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def shrink(epsilon, x):
  y = torch.zeros((x.shape))
  y = y.to(device)

  y[x > epsilon] = -1
  y[x < -epsilon] = +1

  x = x + y * epsilon
  y[y != 0] = 1

  return x.mul(y)
```

For i in 1 to $m \times n$:
  if $S[i] > \lambda$:
    $S[i] = S[i] - \lambda$
    continue
  if $S[i] < -\lambda$:
    $S[i] = S[i] + \lambda$
    continue
  if $-\lambda <= S[i] <= \lambda$:
    $S[i] = 0$
    continue
Return S

```python
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def shrink(epsilon, x):
  y = torch.zeros((x.shape))
  y = y.to(device)

  norm = torch.norm(x, 2, 0)
  for i in range(x.shape[1]):
    if norm[i] > epsilon:
      y[:, i] = x[:,i] - epsilon * x[:,i] / norm[i]
    else:
      y[:, i] = 0.

  return y
```

Input: $S \in R^{m \times n}, \lambda$
For j in 1 to n:
  $e_j = (\sum_{i=1}^{m} |S[i,j]|^2)^{1/2}$
  if $e_j > \lambda$:
    For i in 1 to m:
      $S[i,j] = S[i,j] - \lambda \frac{S[i,j]}{e_j}$
      continue
  if $e_j <= \lambda$:
    For i in 1 to m:
      $S[i,j] = 0$
      continue
Return S

```python
def train(train_data, model, optimizer, EPOCH=10):
  model = model.to(device)

  train_loss = []
  test_loss = []

  for k in range(EPOCH):
    for data in train_data:
      data = data.to(device)
      # compute loss
      data = data.view(-1,28*28).float()

      loss = model.fit(data, optimizer)

      train_loss.append(loss)

  plt.plot(train_loss)
```

```python
def add_noise_uniform_normal(X, corNum=10):
  X = X.view(-1,28*28).float()
  index = torch.randint(0,X.shape[1],(len(X), corNum))
  index = index.to(device)

  for i in range(len(X)):
    X[i, index[i]] = torch.randn(len(index[i])).to(device)
  return X
```

```python
def transform2lable(S):
  x = torch.norm(torch.tensor(S), 0, 1)
  x[torch.where(x == 0)[0]] = 4
  return x

def valid_criterion(pred_label, true_label):
  TP = ((pred_label != 4) & (true_label != 4)).cpu().sum()
  TN = ((pred_label == 4) & (true_label == 4)).cpu().sum()
  FN = ((pred_label == 4) & (true_label != 4)).cpu().sum()
  FP = ((pred_label != 4) & (true_label == 4)).cpu().sum()
  print(TP,TN,FN,FP)
  precision = TP/(TP+FP)
  recall = TP/(TP+FN)
  F1 = 2 * precision * recall / (precision+recall)
  accuracy = (TP+TN) / (TP+TN+FP+FN)

  return [precision, recall, F1, accuracy]
```

UPPSALA
UNIVERSITET

3. Experiments

- Denoising
- Outlier Detection

3.1 Denoising (1/4)

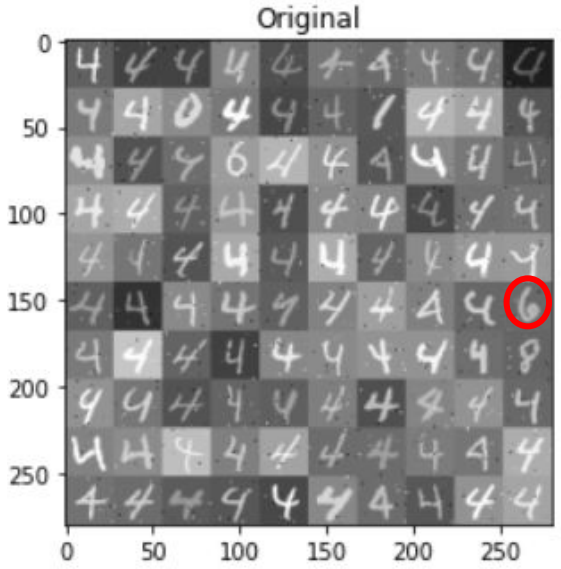
Experiments-Denoising-Experiments Design

- The outstanding advantage of RDAE is that it could denoise an image without a noise-free image.
- To show that, we use a noisy image to train different models. Then compare the results.
  - RDAE,DAE and VAE are trained and compared
- Also, different numbers of noise points are applied in the experiments.
  - 10,100,250 noise points are added in each image (one image has only one digit).

3.1 Denoising (2/4)

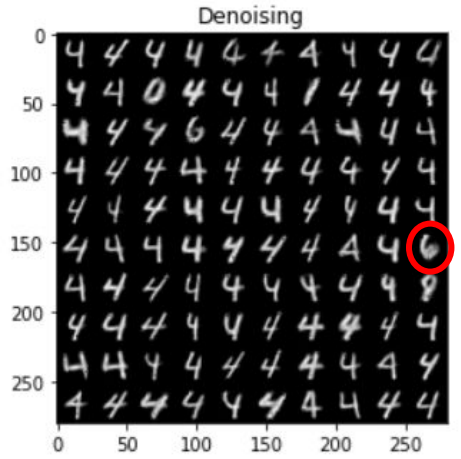
Experiments-Denoising-Results

10 noise points:



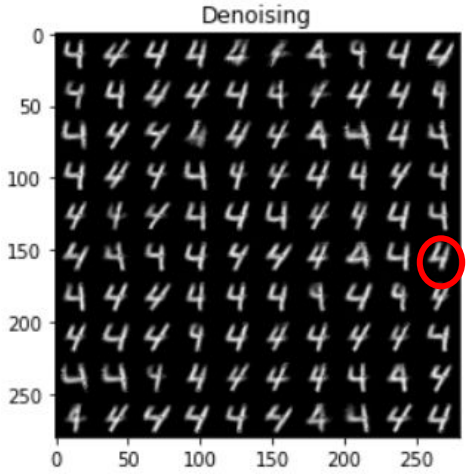
When the noise is small, VAE and RAE can be used for denoising.

RDAE



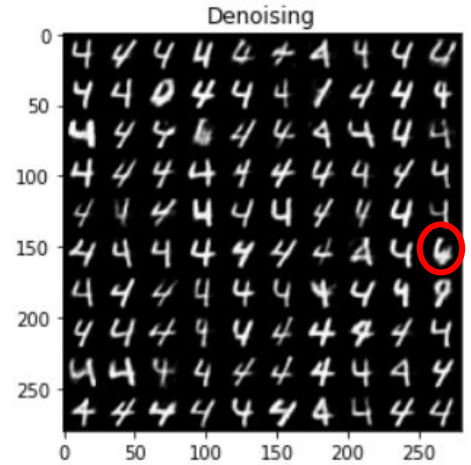
RDAE could reconstruct the digits clearly and denoise the image.

DAE:



DAE could denoise the image, but cannot reconstruct the digits correctly, all digits are transformed to 4, even there are very few noise points.

VAE:



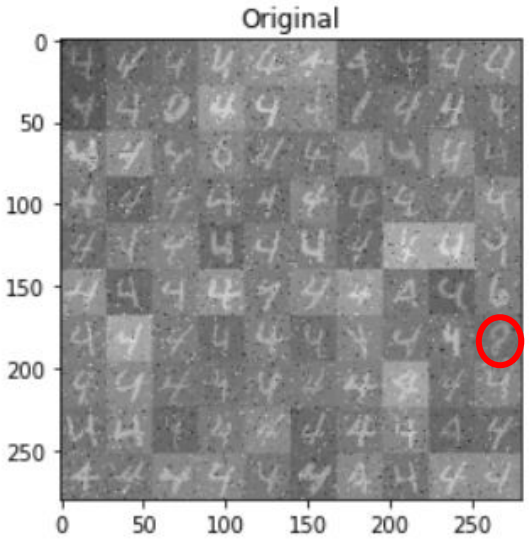
VAE could reconstruct the digits accurately and denoise the image.

UPPSALA UNIVERSITET

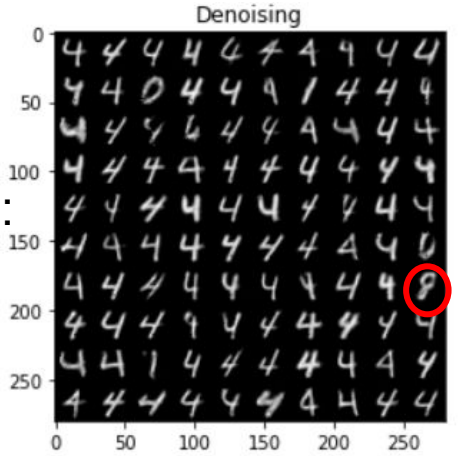
3.1 Denoising (3/4)

Experiments-Denoising-Results

100 noise points:

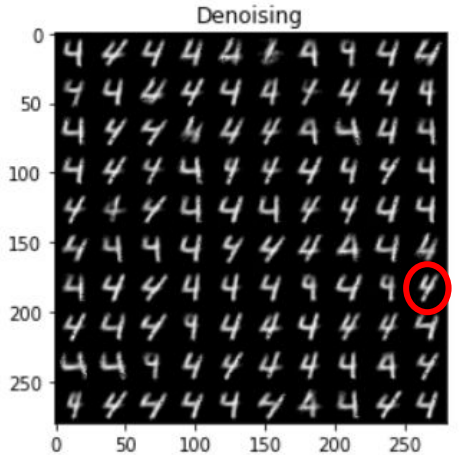


RDAE:



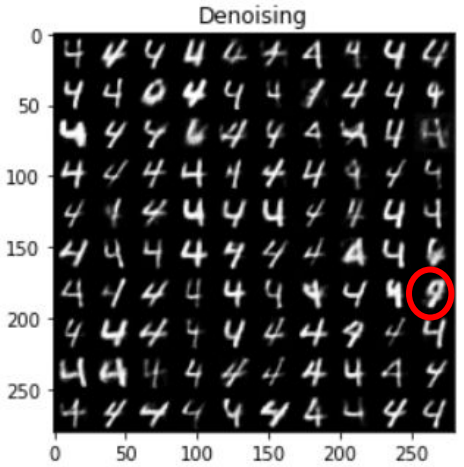
RDAE could reconstruct the digits clearly and denoise the image.

DAE:



DAE could denoise the image, but cannot reconstruct the digits correctly, all digits are transformed to 4.

VAE:



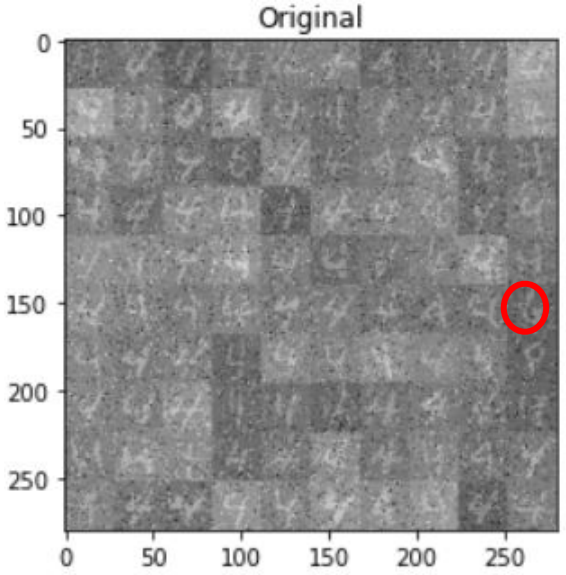
VAE could reconstruct the digits accurately and denoise the image, but not as clear as those of RAE.

UPPSALA UNIVERSITET

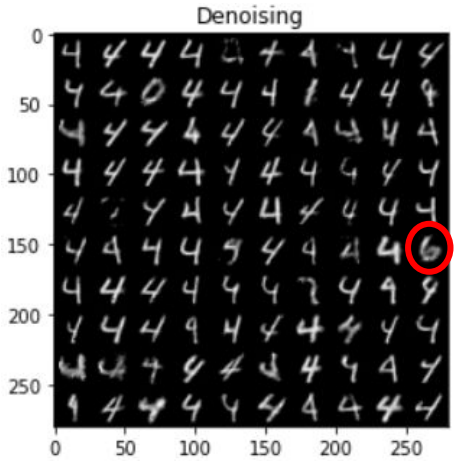
3.1 Denoising (4/4)

Experiments-Denoising-Results

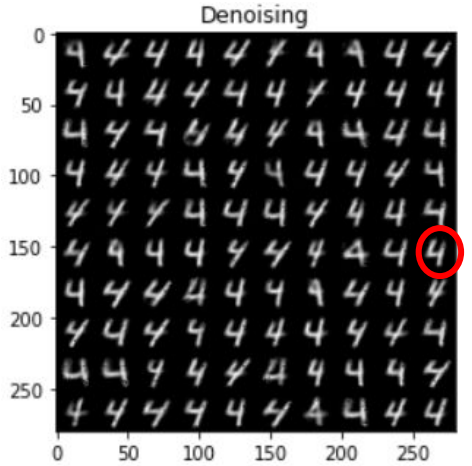
250 noise points:



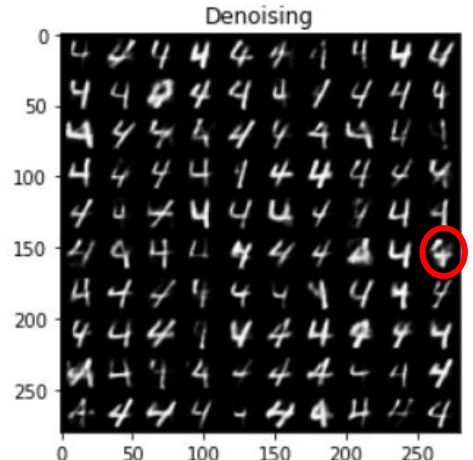
RDAE



DAE:



VAE:

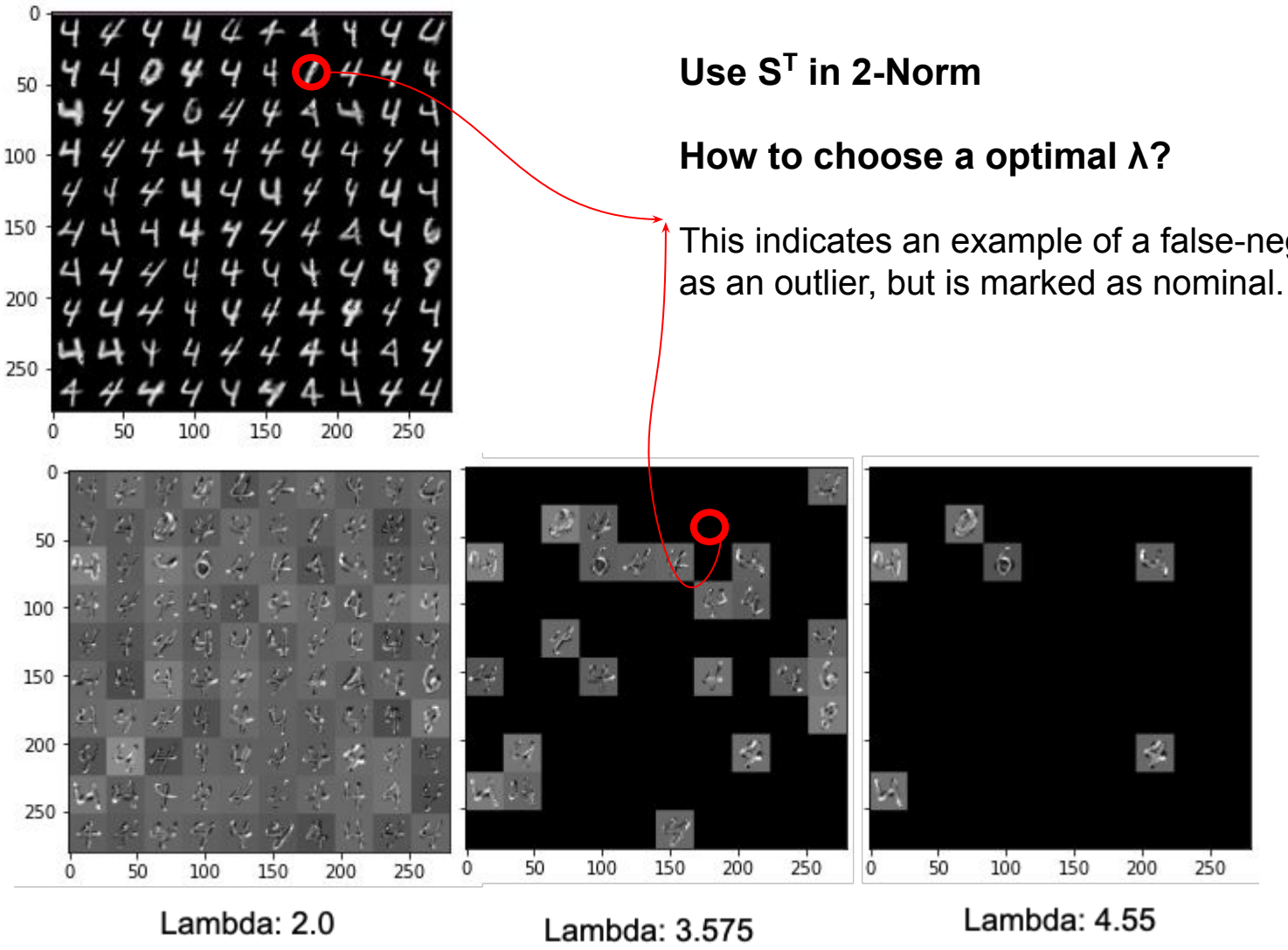


With the number of noise points increasing, the results are nearly the same, but all the reconstructed images become unclear.

When 100 noise points, the VAE could reconstruct 6, but when 250 noise points, it failed to do that. The RAE could still reconstruct 6,8 and other digits.

UPPSALA UNIVERSITET

3.2 Outlier Detection - anomalous instance (1/2)



Use S^T in 2-Norm

How to choose a optimal λ?

This indicates an example of a false-negative; a “1” digit is supposed to be picked out as an outlier, but is marked as nominal.

Since non-zero rows in S are marked as anomalies, the RDA has a high false-positive rate.

When the λ value is larger, placing a heavier penalty on S, and forcing S to be sparser. Many rows are shrunk to zero which reduces the false-positive rate, but also increases the false-negative rate.

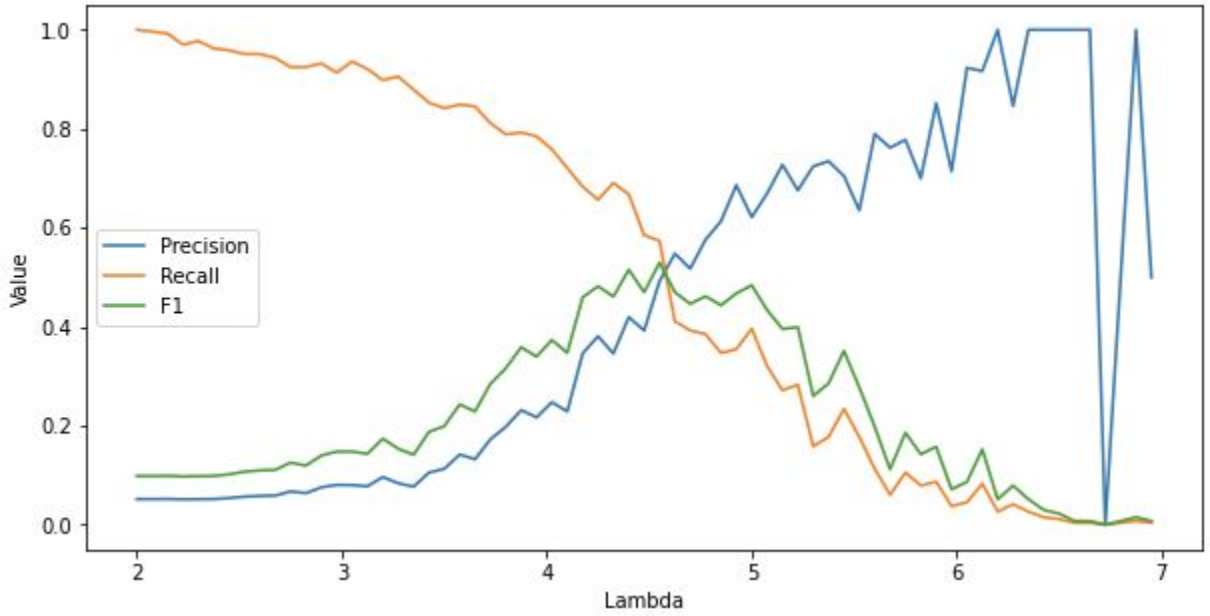
Accordingly, the optimal λ should balance both false-positive and false-negative rates. Thus, we use the F1-score to select the optimal λ

UPPSALA UNIVERSITET

3.2 Outlier Detection - anomalous instance (2/2)

Outlier Detection for l2,1 norm Robust Auto-encoder

First, we try to find the optimal range of lambda. The range is from (2,7, step =0.075)



Here we found the best lambda is around 5, so we pick the values range from (4.5, 5, step = 0.075) and then tune the parameters ‘epoch’ and ‘iteration’.

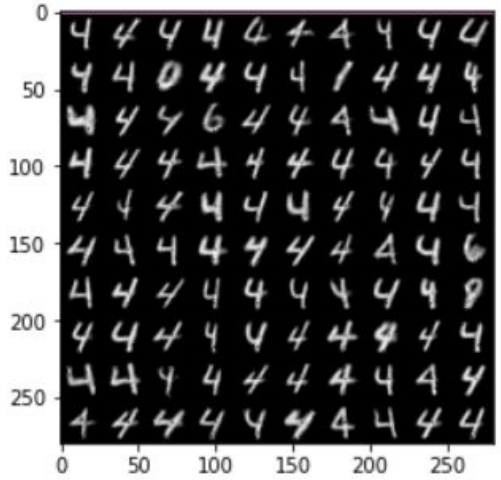
And finally the parameters are set to be: `lambda 4.72500000000000005, epoch 5, iteration 3, with respect to f1 score = 0.6083`

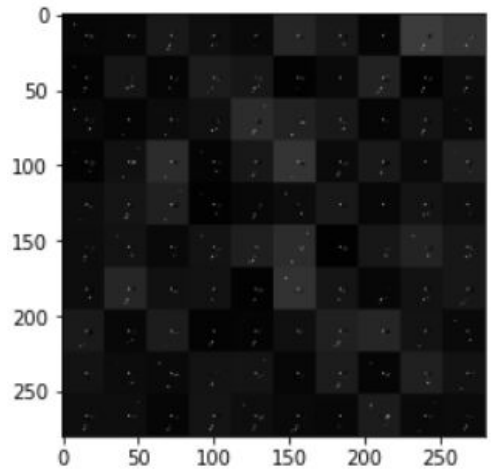
UPPSALA UNIVERSITET

3.2 Outlier Detection - anomalous features (1/1)

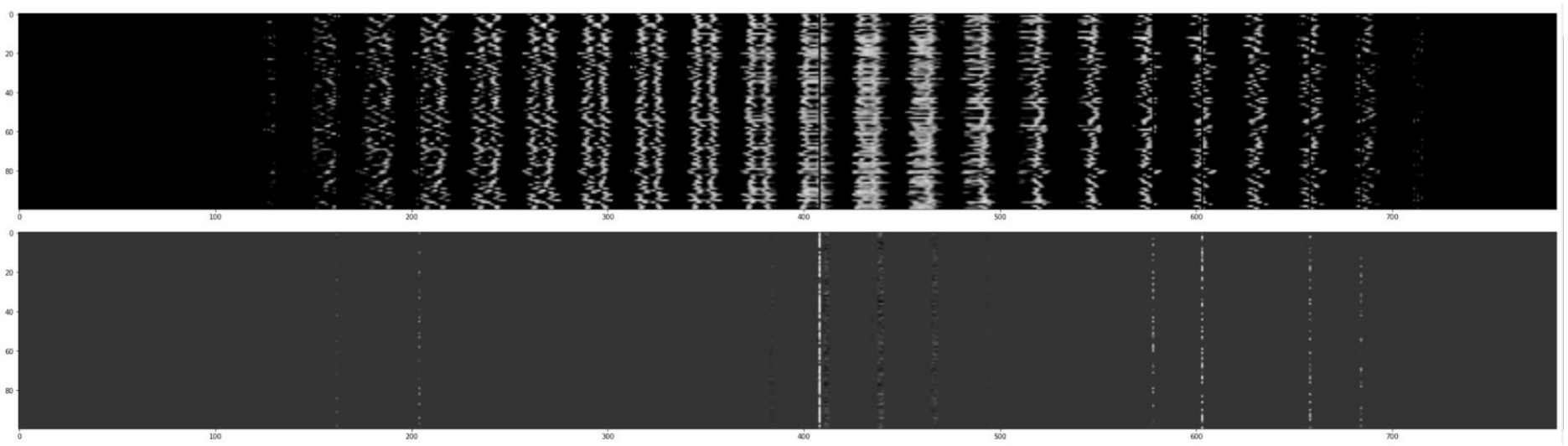
Use S in 2-Norm

view: 28*28





view: 1*784



UPPSALA
UNIVERSITET

Summary

Problems we meet:

- The author used a lot of methods for optimization, which we are not familiar with, for example, projected gradient.
- The algorithm in the article and demo do not match exactly.
- In the demo, the author used a lot of other parameters such as mu, which are not explained at all in the article.
- In the article, the author did not give the method to choose the range of lambda.

Possible improvement:

- Now all the layers in NN are set to be linear. In the future, convolution layer could be applied to see if any improvement.

Thank you for your listening!

UPPSALA
UNIVERSITET