

Code Sample in Python and R

Xinyuan Lyu

February 14, 2025

Introduction

This document contains a collection of code samples from assignments and projects related to **Deep Learning and Time-series Analysis**. The code is written in Python and R, covering various topics in modern machine learning and econometrics.

Contents

1. Deep Learning:

- Linear Regression
- Logistic Regression
- Newton's Method
- Modern Optimization Algorithms
- CNN for Image Detection
- Sparse Autoencoder
- Probabilistic Principal Component Analysis (PPCA)
- Convolutional Autoencoder
- Generative Adversarial Networks (GAN)
- Gaussian-Bernoulli Restricted Boltzmann Machine (GB-RBM)
- Variational Autoencoder (VAE)

2. Time-series Analysis:

- **Stock Market Prediction and Investment Strategies**
 - Augmented Dickey-Fuller (ADF) Test
 - Autoregressive Moving Average (ARMA)
 - Autoregressive Integrated Moving Average (ARIMA)
- **Macroeconomic Analysis of Income Inequality in Japan**
 - Vector Autoregression (VAR)
 - Structural VAR (SVAR)
 - Cointegration Analysis
 - Autoregressive Distributed Lag Model (ARDL)

Homework 1

Xinyuan Lyu

September 2024

Problem 1

```
In [1]: import pandas as pd  
import numpy as np
```

Firstly, load the data and process.

Process the data for (1.)

```
In [2]: df = pd.read_csv('C:/Users/14857/Desktop/ECE685/homework/HW1/code/archive/Concrete_Data_Yeh.csv')  
X = df.iloc[:, :-1].values  
y = df.iloc[:, -1].values  
print(df.head())  
print(X.shape, y.shape)
```

	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

(1030, 8) (1030,)

Process the data for (2.)

```
In [3]: np.random.seed(10)  
indicates = np.arange(X.shape[0])  
np.random.shuffle(indicates) #shuffle the indicates of the data.  
  
split_point = int(X.shape[0] * (1 - 0.25)) #get the split point and split the data randomly to test and train  
X_train, X_test = X[indicates[:split_point]], X[indicates[split_point:]]  
y_train, y_test = y[indicates[:split_point]], y[indicates[split_point:]]  
  
X_trains={}  
X_tests={}  
models = {}  
mse_values = {}  
  
#Generate the data with different dimention of X, i.e. with different number of independent variables.  
for i in range(5, 9):  
    X_trains[i] = X_train[:, :i]  
    X_tests[i] = X_test[:, :i]  
    print(X_trains[i].shape, X_tests[i].shape)  
  
(772, 5) (258, 5)  
(772, 6) (258, 6)  
(772, 7) (258, 7)  
(772, 8) (258, 8)
```

Secondly, construct the model. Using two method.

linear_regression_model_direct_computer uses an analytical solution when solving for parameters, which means it directly calculates them using an explicit optimal formula.

$$L(\beta) = (Y - \hat{Y})^2$$

$$L(\beta) = (Y - \hat{Y})^T(Y - \hat{Y}) = (Y - X\beta)^T(Y - X\beta) = Y^T Y - Y^T X\beta - \beta^T X^T Y + (X\beta)^T X\beta$$

$$= Y^T Y - \beta^T X^T Y - Y^T X\beta + \beta^T X^T X\beta$$

$$\nabla L(\beta) = -2X^T Y + 2X^T X\beta$$

Make $\nabla L(\beta) = 0$ so,

The optimal matrix form for β in linear regression is given by:

$$\beta = (X^T X)^{-1} X^T y$$

```
In [4]: class linear_regression_model_direct_computer:  
  
    def __init__(self, X, y):  
        self.X_matrix = np.hstack([np.ones((X.shape[0], 1)), X]) #Load X and Add the constant term for the model.  
        self.y_vector = y #Load y.  
        self.beta = None #Initialize the beta.  
        self.mse = 0 #Initialize the mse.  
        self.X_test=None #Initialize the test X data.
```

```

self.y_test=None #Initialize the test y data.
self.mse_test=0 #Initialize the test mse.

def compute_beta(self):
    self.beta = np.linalg.inv(self.X_matrix.T @ self.X_matrix) @ self.X_matrix.T @ self.y_vector #Compute the best beta derived from math
    self.y_predict = self.X_matrix @ self.beta #Get the prediction
    return self.beta

def compute_mse(self):
    self.mse = np.mean((self.y_vector - self.X_matrix @ self.beta) ** 2) #compute the mse
    return self.mse

def test_model(self,X_test,y_test):
    self.X_test=np.hstack([np.ones((X_test.shape[0], 1)), X_test]) #Load Test data and Add the constant term for the model.
    self.y_test=y_test
    self.mse_test = np.mean((self.y_test - self.X_test @ self.beta) ** 2) #Compute the mse on test dataset.
    return self.mse_test

```

linear_regression_model_direct_trainer uses Gradient Descend when solving for parameters.

The Gradient of β for mse loss function is given by:

$$\frac{1}{n} (X^T \cdot (X\beta - y))$$

```

In [5]: class linear_regression_model_direct_trainer:

    def __init__(self, X, y, n_epochs, learn_rate):
        self.X_matrix = np.hstack([np.ones((X.shape[0], 1)), X]) #Load X and Add the constant term for the model.
        self.y_vector = y #Load y.
        self.beta = None #Initialize the beta.
        self.y_predict = None #Initialize the preddition.
        self.mse = 0 #Initialize the mse.
        self.X_test=None #Initialize the test X data.
        self.y_test=None #Initialize the test y data.
        self.mse_test=0 #Initialize the test mse.
        self.n_epochs=n_epochs #Set number of epochs
        self.learn_rate=learn_rate #Set learning rate

    def compute_beta(self):
        self.beta = np.zeros(self.X_matrix.shape[1]) #Initialize the beta.
        n=self.X_matrix.shape[0] #get the n which is the sample size.

        for epoch in range(self.n_epochs):
            loss = np.mean((self.X_matrix @ self.beta - self.y_vector) ** 2) #Define the Loss function
            self.beta = self.beta - self.learn_rate*(1/n)*(self.X_matrix.T @ (self.X_matrix @ self.beta - self.y_vector)) #The gradient descent
            print(loss)
        return self.beta

    def compute_mse(self):
        self.mse = np.mean((self.y_vector - self.X_matrix @ self.beta) ** 2)
        return self.mse

    def test_model(self,X_test,y_test):
        self.X_test=np.hstack([np.ones((X_test.shape[0], 1)), X_test])
        self.y_test=y_test
        self.mse_test = np.mean((self.y_test - self.X_test @ self.beta) ** 2)
        return self.mse_test

```

Thirdly, use the data and model to solve (1.) and (2.).

For (1.), use two model defined as above. And get the beta as well as MSE.

```

In [6]: model_with_all_independent_variables_compute=linear_regression_model_direct_computer(X,y) # Use the full data to train the model.
print(f'The optimal beta {model_with_all_independent_variables_compute.compute_beta()}')
print(f'The final mse {model_with_all_independent_variables_compute.compute_mse()}')

The optimal beta [-2.33312136e+01  1.19804334e-01  1.03865809e-01  8.79343215e-02
-1.49918419e-01  2.92224595e-01  1.80862148e-02  2.01903511e-02
1.14222068e-01]
The final mse 107.19723607486017

```

```

In [7]: %%capture
model_with_all_independent_variables_train=linear_regression_model_direct_trainer(X,y,100000,0.000001) #100000,0.000001 are number of epochs
model_with_all_independent_variables_train.compute_beta()

```

```

In [8]: print(f'The optimal beta {model_with_all_independent_variables_train.beta}')
print(f'The final mse {model_with_all_independent_variables_train.compute_mse()}')

The optimal beta [-0.00026361  0.11510396  0.09830647  0.08285372 -0.18885138  0.20432239
 0.0100605  0.01256648  0.11421329]
The final mse 107.32757392939513

```

We can see that Model use Gradient Descend may has the mse close to Model use analytical solution, but cannot be same as it.

For (2.), we choose i=6,7,8,9 (including the constant term). We may also use two models.

```
In [9]: for i in range(5, 9):
    X_train = X_trains[i]
    X_test = X_tests[i]
    models[i] = linear_regression_model_direct_computer(X_train, y_train)
    models[i].compute_beta()
    mse_values[i]=models[i].test_model(X_test,y_test)

print(f"Model with 6 characters (including constant term) MSE: {mse_values[5]}")
print(f"Model with 7 characters (including constant term) MSE: {mse_values[6]}")
print(f"Model with 8 characters (including constant term) MSE: {mse_values[7]}")
print(f"Model with 9 characters (including constant term) MSE: {mse_values[8]}")
```

```
Model with 6 characters (including constant term) MSE: 159.64197170664195
Model with 7 characters (including constant term) MSE: 159.1847643187316
Model with 8 characters (including constant term) MSE: 159.19128766119093
Model with 9 characters (including constant term) MSE: 108.20657336621308
```

```
In [10]: %%capture
for i in range(5, 9):
    X_train = X_trains[i]
    X_test = X_tests[i]
    models[i] = linear_regression_model_direct_trainer(X_train, y_train,100000,0.000001)
    models[i].compute_beta()
    mse_values[i]=models[i].test_model(X_test,y_test)
```

```
In [11]: print(f"Model with 6 characters (including constant term) MSE: {mse_values[5]}")
print(f"Model with 7 characters (including constant term) MSE: {mse_values[6]}")
print(f"Model with 8 characters (including constant term) MSE: {mse_values[7]}")
print(f"Model with 9 characters (including constant term) MSE: {mse_values[8]}")
```

```
Model with 6 characters (including constant term) MSE: 160.69504467726168
Model with 7 characters (including constant term) MSE: 157.51131666603607
Model with 8 characters (including constant term) MSE: 158.89061737211492
Model with 9 characters (including constant term) MSE: 107.87370194419836
```

From the result we can get the conclusion that:

- (1) Models with 6,7,8 characters (including constant term) have very close MSE, and even increasing.
- (2) And Model from 8 to 9 have a large descend in MSE.

So it shows that:

It is Not the case that models with more independent variables always perform better.

From (2) I think that character 9 is very important for impacting or predicting y than other variables like 7,8.

And From (1) I think that the increase in MSE suggests that 7,8 may be irrelevant features and cause overfitting or noise for model and make model perform worse.

This highlights the importance of selecting key features rather than simply increasing the number of features.

End of Problem 1.

```
In [12]: %%reset -f
```

Problem 2

```
In [13]: import torch
import torchvision
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
from IPython import display
from torchvision import datasets, transforms
from torch.utils import data
from torch.utils.data import DataLoader, Dataset, TensorDataset

from Encoder import extractor # import the extractor which is given.
```

Firstly, we load the extractor as the (1.) required.

```
In [14]: %%capture
from Encoder import extractor

my_extractor=extractor()
extractor_weights = 'C:/Users/14857/Desktop/ECE685/homework/HW1/code/feature_extractor_weights.pth'
```

```
my_extractor.load_state_dict(torch.load(extractor_weights))
my_extractor.eval()
```

Next, we download the MNIST data and transform it to tensor as well as normalizing the data.

```
In [15]: transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))

])
Batch_size=32
train_data = torchvision.datasets.MNIST("./data", train=True, download=True, transform=transform)
test_data = torchvision.datasets.MNIST("./data", train=False, download=True, transform=transform)
train_data_loader = DataLoader(train_data, batch_size=Batch_size, shuffle=True)
test_data_loader = DataLoader(test_data, batch_size=Batch_size, shuffle=False)
```

Secondly, we use the extractor to extract the latent representations as (2.) required and generate the datasets for the next step of training.

```
In [16]: train_features = [] #set empty list.
train_labels = []
test_features = []
test_labels = []

with torch.no_grad():
    for images, label in train_data_loader:
        feature = my_extractor(images)
        train_features.append(feature)
        train_labels.append(label)

train_features = torch.cat(train_features, dim=0) #combine the tensors i.e. make the list to a whole tensor.
train_labels = torch.cat(train_labels, dim=0) #combine the tensors i.e. make the list to a whole tensor.

with torch.no_grad():
    for images, label in test_data_loader:
        feature = my_extractor(images)
        test_features.append(feature)
        test_labels.append(label)

test_features = torch.cat(test_features, dim=0) #combine the tensors i.e. make the list to a whole tensor.
test_labels = torch.cat(test_labels, dim=0) #combine the tensors i.e. make the list to a whole tensor.
```

Packaging the features tensor and labels tensor to dataset. And then transfor to the dataloader.

```
In [17]: extracted_train_data = TensorDataset(train_features, train_labels)
extracted_test_data = TensorDataset(test_features, test_labels)

Batch_size=32
extracted_train_data_loader = DataLoader(extracted_train_data, batch_size=Batch_size, shuffle=True) # Shuffle is true for the SGD later.
extracted_test_data_loader = DataLoader(extracted_test_data, batch_size=Batch_size, shuffle=False)
```

Thirdly, define the model, softmax function, cross entropy loss function and the gradient of loss respect to W and B, as (3.) required.

Softmax Regression Gradient Derivation, where y is the one-hot vector of labels of images, and y_k means the k^{th} entry of y .

$$\mathbb{P}(Y|H, w_1, \dots, w_k, b) = \prod_{n=1}^N \prod_{k=1}^K \mathbb{P}(C_k|h_n)^{y_{nk}} = \prod_{n=1}^N \prod_{k=1}^K \hat{y}_{nk}^{y_{nk}}$$

where

$$\hat{y}_{nk} = \mathbb{P}(C_k|h_n) = \frac{\exp(w_k^T h_n + b)}{\sum_j \exp(w_j^T h_n + b)}$$

The loss function $E(w_1, \dots, w_k, b)$ is the negative log-likelihood:

$$\begin{aligned} E(w_1, \dots, w_k, b) &= -\ln \mathbb{P}(Y|H, w_1, \dots, w_k, b) \\ &= -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \ln \hat{y}_{nk} \\ &= -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \ln \left(\frac{\exp(w_k^T h_n + b)}{\sum_j \exp(w_j^T h_n + b)} \right) \\ &= -\sum_{n=1}^N \sum_{k=1}^K \left[y_{nk} \left(w_k^T h_n + b - \ln \sum_j \exp(w_j^T h_n + b) \right) \right] \end{aligned}$$

$$= \sum_{n=1}^N \left[\ln \sum_j \exp(w_j^T h_n + b) - \sum_{k=1}^K y_{nk}(w_k^T h_n + b) \right]$$

So the average loss of every sample point is

$$= \frac{1}{N} \cdot \sum_{n=1}^N \left[\ln \sum_j \exp(w_j^T h_n + b) - \sum_{k=1}^K y_{nk}(w_k^T h_n + b) \right]$$

For the gradient w.r.t. (w_i):

$$\begin{aligned} \nabla_{AvgE_{w_i}} &= \frac{1}{N} \sum_{n=1}^N \left(\frac{\exp(w_i^T h_n + b)}{\sum_j \exp(w_j^T h_n + b)} h_n - y_{ni} h_n \right) \\ &= \frac{1}{N} \sum_{n=1}^N ((\hat{y}_{ni} - y_{ni}) h_n) \end{aligned}$$

For the gradient w.r.t. (b_i):

$$\begin{aligned} \nabla_{AvgE_{b_i}} &= \frac{1}{N} \sum_{n=1}^N \left(\frac{\exp(w_i^T h_n + b)}{\sum_j \exp(w_j^T h_n + b)} - y_{ni} \right) \\ &= \frac{1}{N} \sum_{n=1}^N (\hat{y}_{ni} - y_{ni}) \end{aligned}$$

So we can write the code to realize the math derivation.

```
In [18]: def softmax(X): # Define the softmax function as the definition.
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdim=True)
    return X_exp / partition

def model(X,W,b): # Define the softmax regression model.
    return softmax(torch.matmul(X, W) + b)

def to_one_hot(y,num_classes): # Define the transformation function for y to the one hot code of y.
    return torch.nn.functional.one_hot(y, num_classes=num_classes)

def cross_entropy(y_hat, y): # Define the cross entropy loss function.
    return - torch.log(y_hat[range(len(y_hat)), y])

def loss(y_hat,y): # So the Loss function is the sum of cross entropy from 1 to n.
    return (cross_entropy(y_hat, y).sum()) / Batch_size

def gradient_W(X,y_hat,y_one_hot): # The gradient w.r.t. w is as derived.
    return (torch.matmul(X.T, (y_hat - y_one_hot))) / Batch_size

def gradient_b(X,y_hat,y_one_hot): # The gradient w.r.t. b is as derived.
    return ((y_hat - y_one_hot).sum(dim=0)) / Batch_size
```

Initialize the weight and bias, and set the learning rate and number of epochs, and use GPU to train the model.

```
In [19]: num_inputs = 256 # The dimension of the X i.e. the features of the images.
num_outputs = 10 # The dimension of the one-hot vector of y i.e. y has 10 classes.
learn_rate = 0.001 # set the learning rate.
num_epochs = 50

W = torch.normal(0, 0.01, size=(num_inputs, num_outputs), requires_grad=True)
b = torch.zeros(size=(1, num_outputs), requires_grad=True)

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

train_losses = [] # Prepare for the plotting later.
test_losses = []
train_accuracies = []
test_accuracies = []
```

Forthly, train the data and plot the loss as well as accuracy in train data and test data for each epoch, as the (4.) required.

And we will show two method to optimize the model W and b , the First is based on the math and the gradients obtained in (3.), and the Second is based on the Automatic Differentiation function in PyTorch.

(a.) The First method, manually derive the Gradient and then train the model i.e. from scratch.

```
In [20]: for epoch in range(num_epochs):
    total_train_loss = 0 #initialize the loss and accuracy in each epoch.
    correct_train = 0
    total_train = 0

    for i, data in enumerate(extracted_train_data_loader):
        X, y = data
        X = X.to(device)
        y = y.to(device)

        if X.shape != (Batch_size, num_inputs): # Avoid the shape not matching.
            break

        y_one_hot = to_one_hot(y, 10).to(device)
        y_hat = model(X, W, b).to(device)

        ls_train = loss(y_hat, y)
        total_train_loss += ls_train.item()

        W = W - learn_rate * gradient_W(X, y_hat, y_one_hot)
        b = b - learn_rate * gradient_b(X, y_hat, y_one_hot)

        predicted_train = y_hat.argmax(dim=1)
        correct_train += (predicted_train == y).sum().item()
        total_train += y.size(0)

    average_train_loss = total_train_loss / len(extracted_train_data_loader)
    train_losses.append(average_train_loss)
    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy)

# In each epoch, we also test the model in test data.
with torch.no_grad():

    total_test_loss = 0
    correct_test = 0
    total_test = 0

    for i, data in enumerate(extracted_test_data_loader):
        X_test, y_test = data
        X_test = X_test.to(device)
        y_test = y_test.to(device)

        if X_test.shape != (Batch_size, num_inputs):
            break

        y_hat_test = model(X_test, W, b)
        test_loss = loss(y_hat_test, y_test)

        total_test_loss += test_loss.item()
        predicted_test = y_hat_test.argmax(dim=1)
        correct_test += (predicted_test == y_test).sum().item()
        total_test += y_test.size(0)

    average_test_loss = total_test_loss / len(extracted_test_data_loader)
    test_losses.append(average_test_loss)
    test_accuracy = correct_test / total_test
    test_accuracies.append(test_accuracy)

if epoch % 5 == 0: # every 5 epochs print the loss and accuracy.
    print(f'Epoch {epoch+1}, Train Loss: {average_train_loss:.4f}, Test Loss: {average_test_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}')



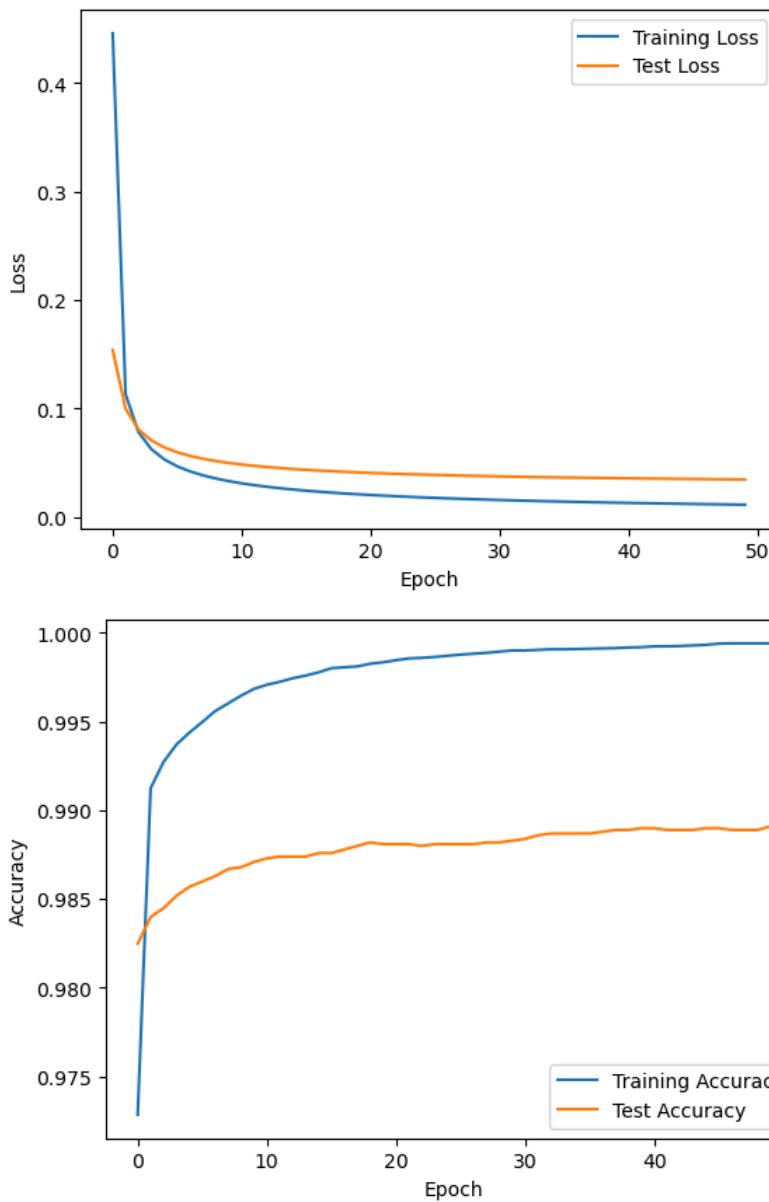
```

Epoch 1, Train Loss: 0.4460, Test Loss: 0.1539, Train Accuracy: 0.9728, Test Accuracy: 0.9825
Epoch 6, Train Loss: 0.0466, Test Loss: 0.0596, Train Accuracy: 0.9950, Test Accuracy: 0.9860
Epoch 11, Train Loss: 0.0309, Test Loss: 0.0483, Train Accuracy: 0.9971, Test Accuracy: 0.9873
Epoch 16, Train Loss: 0.0241, Test Loss: 0.0434, Train Accuracy: 0.9980, Test Accuracy: 0.9876
Epoch 21, Train Loss: 0.0202, Test Loss: 0.0405, Train Accuracy: 0.9984, Test Accuracy: 0.9881
Epoch 26, Train Loss: 0.0175, Test Loss: 0.0387, Train Accuracy: 0.9988, Test Accuracy: 0.9881
Epoch 31, Train Loss: 0.0155, Test Loss: 0.0373, Train Accuracy: 0.9990, Test Accuracy: 0.9884
Epoch 36, Train Loss: 0.0140, Test Loss: 0.0363, Train Accuracy: 0.9991, Test Accuracy: 0.9887
Epoch 41, Train Loss: 0.0129, Test Loss: 0.0355, Train Accuracy: 0.9992, Test Accuracy: 0.9890
Epoch 46, Train Loss: 0.0119, Test Loss: 0.0349, Train Accuracy: 0.9994, Test Accuracy: 0.9890

Plot the loss and accuracy for each epoch in the training process.

```
In [21]: def plot_the_model(y1, y2, type='loss'):
    plt.figure()
    if type == 'loss':
        plt.plot(y1, label='Training Loss')
        plt.plot(y2, label='Test Loss')
        plt.ylabel('Loss')
    elif type == 'accuracy':
        plt.plot(y1, label='Training Accuracy')
        plt.plot(y2, label='Test Accuracy')
        plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()
```

```
plot_the_model(train_losses, test_losses, type='loss')
plot_the_model(train_accuracies, test_accuracies, type='accuracy')
```



(b.) The Second method, use the package in pytorch and use SGD to train the model.

```
In [22]: W = torch.normal(0, 0.01, size=(num_inputs, num_outputs), requires_grad=True, device=device)
b = torch.zeros(size=(1, num_outputs), requires_grad=True, device=device)
optimizer = optim.SGD([W, b], lr=learn_rate, weight_decay=0.001) # Define the optimizer. And we set weight_decay=0.001 to avoid the overfitting
```

Others are all the same except the optimizer.

```
In [23]: train_losses = [] # Prepare for the plotting later.
test_losses = []
train_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):

    total_train_loss = 0 #initialize the loss and accuracy in each epoch.
    correct_train = 0
    total_train = 0

    for i, data in enumerate(extracted_train_data_loader):

        X, y = data
        X = X.to(device)
        y = y.to(device)

        if X.shape != (Batch_size, num_inputs): # Avoid the shape not matching.
            break

        y_one_hot = to_one_hot(y, 10).to(device)
        y_hat = model(X, W, b).to(device)

        ls_train = loss(y_hat, y)
        total_train_loss += ls_train.item()

        optimizer.zero_grad() #Use the optimizer.
        ls_train.backward()
```

```

optimizer.step()

predicted_train = y_hat.argmax(dim=1)
correct_train += (predicted_train == y).sum().item()
total_train += y.size(0)

average_train_loss = total_train_loss / len(extracted_train_data_loader)
train_losses.append(average_train_loss)
train_accuracy = correct_train / total_train
train_accuracies.append(train_accuracy)

# In each epoch, we also test the model in test data.
with torch.no_grad():

    total_test_loss = 0
    correct_test = 0
    total_test = 0

    for i, data in enumerate(extracted_test_data_loader):

        X_test, y_test = data
        X_test = X_test.to(device)
        y_test = y_test.to(device)

        if X_test.shape != (Batch_size, num_inputs):
            break

        y_hat_test = model(X_test, W, b)
        test_loss = loss(y_hat_test, y_test)

        total_test_loss += test_loss.item()
        predicted_test = y_hat_test.argmax(dim=1)
        correct_test += (predicted_test == y_test).sum().item()
        total_test += y_test.size(0)

    average_test_loss = total_test_loss / len(extracted_test_data_loader)
    test_losses.append(average_test_loss)
    test_accuracy = correct_test / total_test
    test_accuracies.append(test_accuracy)

if epoch % 5 == 0: # every 5 epochs print the loss and accuracy.
    print(f'Epoch {epoch+1}, Train Loss: {average_train_loss:.4f}, Test Loss: {average_test_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}')

```

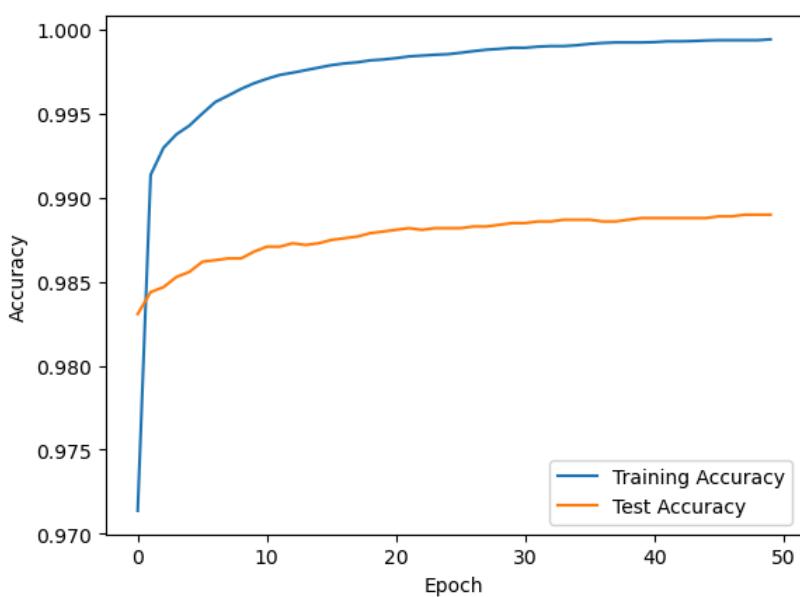
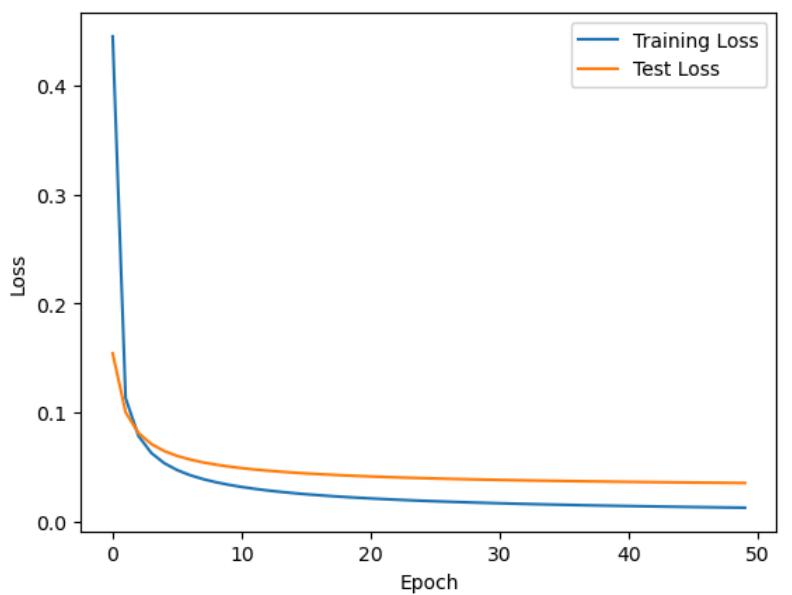
Epoch 1, Train Loss: 0.4452, Test Loss: 0.1543, Train Accuracy: 0.9714, Test Accuracy: 0.9831
Epoch 6, Train Loss: 0.0471, Test Loss: 0.0601, Train Accuracy: 0.9950, Test Accuracy: 0.9862
Epoch 11, Train Loss: 0.0316, Test Loss: 0.0489, Train Accuracy: 0.9971, Test Accuracy: 0.9871
Epoch 16, Train Loss: 0.0249, Test Loss: 0.0440, Train Accuracy: 0.9979, Test Accuracy: 0.9875
Epoch 21, Train Loss: 0.0211, Test Loss: 0.0412, Train Accuracy: 0.9983, Test Accuracy: 0.9881
Epoch 26, Train Loss: 0.0185, Test Loss: 0.0394, Train Accuracy: 0.9986, Test Accuracy: 0.9882
Epoch 31, Train Loss: 0.0166, Test Loss: 0.0381, Train Accuracy: 0.9989, Test Accuracy: 0.9885
Epoch 36, Train Loss: 0.0152, Test Loss: 0.0371, Train Accuracy: 0.9991, Test Accuracy: 0.9887
Epoch 41, Train Loss: 0.0141, Test Loss: 0.0363, Train Accuracy: 0.9992, Test Accuracy: 0.9888
Epoch 46, Train Loss: 0.0132, Test Loss: 0.0357, Train Accuracy: 0.9994, Test Accuracy: 0.9889

```

In [24]: def plot_the_model(y1, y2, type='loss'):
    plt.figure()
    if type == 'loss':
        plt.plot(y1, label='Training Loss')
        plt.plot(y2, label='Test Loss')
        plt.ylabel('Loss')
    elif type == 'accuracy':
        plt.plot(y1, label='Training Accuracy')
        plt.plot(y2, label='Test Accuracy')
        plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()

plot_the_model(train_losses, test_losses, type='loss')
plot_the_model(train_accuracies, test_accuracies, type='accuracy')

```



From the data on the loss function and accuracy on both the training set and test set, we can see that our model exhibits good properties, achieving an accuracy of up to 98% on the test set. Additionally, it is evident that both optimization methods—manual gradient calculation and manual implementation of gradient descent, as well as using PyTorch's automatic differentiation and optimization—are able to effectively train the model. The difference lies in the fact that the second method is more concise and maybe allows the addition of regularization to mitigate overfitting.

End of Problem 2.

Problem 3

I will give two solutions for 1, the first is based on the change of variables rule and the other is based on the pdf of $Z=XY$ where X, Y is independent.

1.

Firstly, we have:

$$\begin{aligned} F_R(r) &= \mathbb{P}(R \leq r) = \mathbb{P}(R^2 \leq r^2) \quad (\text{since } R \geq 0) \\ &= \mathbb{P}(-2 \ln U_2 \leq r^2) = \mathbb{P}(\ln U_2 \geq -\frac{r^2}{2}) = \mathbb{P}(U_2 \geq e^{-\frac{r^2}{2}}) \\ &= 1 - \mathbb{P}(U_2 \leq e^{-\frac{r^2}{2}}) = 1 - \exp\left(-\frac{r^2}{2}\right) \\ \text{So } f_R(r) &= F'_R(r) = r \cdot \exp\left(-\frac{r^2}{2}\right), \quad r \in (0, +\infty) \end{aligned}$$

Let $\Theta = 2\pi \cdot U_1$, so $\Theta \sim U(0, 2\pi)$, so

$$F_\Theta(\theta) = \mathbb{P}(\Theta \leq \theta) = \frac{\theta}{2\pi}, \quad \theta \in (0, 2\pi)$$

$$\text{So } f_\Theta(\theta) = F'_\Theta(\theta) = \frac{1}{2\pi}$$

And We have:

$$R \perp\!\!\!\perp \Theta \quad (\text{since } U_1 \perp\!\!\!\perp U_2)$$

So

$$f_{R,\Theta}(r, \theta) = f_R(r) \cdot f_\Theta(\theta)$$

And we have the transformation from (R, Θ) to (Z_1, Z_2) :

$$Z_1 = R \cos \Theta \quad Z_2 = R \sin \Theta$$

Thus

$$R = \sqrt{Z_1^2 + Z_2^2} \quad \Theta = \arctan\left(\frac{Z_2}{Z_1}\right)$$

So for the Jacobian determinant, we have: Jacobian matrix J is defined as:

$$J = \frac{\partial(R, \Theta)}{\partial(Z_1, Z_2)} = \begin{pmatrix} \frac{\partial R}{\partial Z_1} & \frac{\partial R}{\partial Z_2} \\ \frac{\partial \Theta}{\partial Z_1} & \frac{\partial \Theta}{\partial Z_2} \end{pmatrix}$$

And we can compute each term:

$$\begin{aligned}\frac{\partial R}{\partial Z_1} &= \frac{Z_1}{\sqrt{Z_1^2 + Z_2^2}}, & \frac{\partial R}{\partial Z_2} &= \frac{Z_2}{\sqrt{Z_1^2 + Z_2^2}} \\ \frac{\partial \Theta}{\partial Z_1} &= -\frac{Z_2}{Z_1^2 + Z_2^2}, & \frac{\partial \Theta}{\partial Z_2} &= \frac{Z_1}{Z_1^2 + Z_2^2}\end{aligned}$$

Thus the Jacobian determinant is

$$J = \begin{pmatrix} \frac{Z_1}{\sqrt{Z_1^2 + Z_2^2}} & \frac{Z_2}{\sqrt{Z_1^2 + Z_2^2}} \\ -\frac{Z_2}{Z_1^2 + Z_2^2} & \frac{Z_1}{Z_1^2 + Z_2^2} \end{pmatrix}$$

Therefore

$$|det(J)| = \frac{1}{\sqrt{Z_1^2 + Z_2^2}}$$

So use the Change of the variables rule:

$$\begin{aligned}f_{Z_1, Z_2}(z_1, z_2) &= f_{R, \Theta} \left(\sqrt{z_1^2 + z_2^2}, \arctan \left(\frac{z_2}{z_1} \right) \right) \cdot |det(J)| \\ &= \frac{1}{2\pi} \exp \left(-\frac{z_1^2 + z_2^2}{2} \right) \cdot \sqrt{z_1^2 + z_2^2} \cdot \left(\frac{1}{\sqrt{z_1^2 + z_2^2}} \right) \\ &= \frac{1}{2\pi} \exp \left(-\frac{z_1^2 + z_2^2}{2} \right)\end{aligned}$$

So for Z_1 and Z_2 we have the marginal distribution:

Therefore

$$f_{Z_1}(z_1) = \int_{-\infty}^{\infty} f_{Z_1, Z_2}(z_1, z_2) dz_2 = \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{z_1^2}{2} \right) \cdot \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{z_2^2}{2} \right) dz_2$$

And from the property of normal distribution, we know that

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{x^2}{2} \right) dx = 1$$

So

$$f_{Z_1}(z_1) = \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{z_1^2}{2} \right)$$

Similarly, we have:

$$f_{Z_2}(z_2) = \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{z_2^2}{2} \right)$$

This is exactly the probability density function of the standard normal distribution.
Thus

$$Z_1 \sim \mathcal{N}(0, 1) \quad \text{and} \quad Z_2 \sim \mathcal{N}(0, 1) \quad \square$$

2.

From 1. we have:

$$f_{Z_1, Z_2}(z_1, z_2) = \frac{1}{2\pi} \exp\left(-\frac{z_1^2 + z_2^2}{2}\right)$$

And we have:

$$f_{Z_1}(z_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_1^2}{2}\right)$$

$$f_{Z_2}(z_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_2^2}{2}\right)$$

So it is easy to see that:

$$f_{Z_1, Z_2}(z_1, z_2) = f_{Z_1}(z_1) \cdot f_{Z_2}(z_2)$$

Thus, this means that Z_1 and Z_2 are independent. \square

3. The Alternative Solution for 1.

We can also derive the pdf of Z1 and Z2 directly using the fact that R and Θ are independent.

Firstly, we have:

$$\begin{aligned} F_R(r) &= \mathbb{P}(R \leq r) = \mathbb{P}(R^2 \leq r^2) \quad (\text{since } R \geq 0) \\ &= \mathbb{P}(-2 \ln U_2 \leq r^2) = \mathbb{P}(\ln U_2 \geq -\frac{r^2}{2}) = \mathbb{P}(U_2 \geq e^{-\frac{r^2}{2}}) \\ &= 1 - \mathbb{P}(U_2 \leq e^{-\frac{r^2}{2}}) = 1 - \exp\left(-\frac{r^2}{2}\right) \\ \text{So } f_R(r) &= F'_R(r) = r \cdot \exp\left(-\frac{r^2}{2}\right), \quad r \in (0, +\infty) \end{aligned}$$

Let $X_1 = \cos \Theta$, $X_2 = \sin \Theta$, then

$$\begin{aligned} F_{X_1}(x) &= \mathbb{P}(X_1 \leq x) = \mathbb{P}(\cos(2\pi U_1) \leq x) \\ &= \mathbb{P}(\arccos x \leq 2\pi U_1 \leq 2\pi - \arccos x), \quad x \in (-1, 1) \\ &= \mathbb{P}\left(\frac{\arccos x}{2\pi} \leq U_1 \leq 1 - \frac{\arccos x}{2\pi}\right) \\ &= 1 - \frac{\arccos x}{\pi}, \quad x \in (-1, 1) \end{aligned}$$

$$\text{So } f_{X_1}(x) = F'_{X_1}(x) = \left(-\frac{1}{\pi} \cdot -\frac{1}{\sqrt{1-x^2}}\right) = \frac{1}{\pi \sqrt{1-x^2}}, \quad x \in (-1, 1)$$

Similarly,

$$\begin{aligned} F_{X_2}(x) &= \mathbb{P}(X_2 \leq x) = \mathbb{P}(\sin(2\pi U_1) \leq x) \\ &= \mathbb{P}(-\pi - \arcsin x \leq 2\pi U_1 \leq \arcsin x) \\ &= \mathbb{P}\left(-\frac{1}{2} - \frac{\arcsin x}{2\pi} \leq U_1 \leq \frac{\arcsin x}{2\pi}\right) \\ &= \frac{1}{2} + \frac{\arcsin x}{\pi}, \quad x \in (-1, 1) \end{aligned}$$

$$\text{So } f_{X_2}(x) = F'_{X_2}(x) = \frac{1}{\pi \sqrt{1-x^2}}, \quad x \in (-1, 1)$$

Thus,

$$f_{X_2}(x) = f_{X_1}(x) = \frac{1}{\pi \sqrt{1-x^2}} \quad [1]$$

And We have:

$$R \perp\!\!\!\perp X_1 \text{ and } R \perp\!\!\!\perp X_2 \quad (\text{since } U_1 \perp\!\!\!\perp U_2) \quad [2]$$

Therefore we have:

$$Z_1 = RX_1 \implies X_1 = \frac{Z_1}{R} \in (-1, 1), \text{ so } R > |Z_1|$$

$$Z_2 = RX_2 \implies X_2 = \frac{Z_2}{R} \in (-1, 1), \text{ so } R > |Z_2| \quad [3]$$

Due to [1] ; [2] and [3], we get:

$$\begin{aligned} f_{Z_2}(z) &= f_{Z_1}(z) = \int_{|z|}^{\infty} \frac{1}{|r|} f_{R, X_1}(r, z/r) dr \\ &= \int_{|z|}^{\infty} \frac{1}{r} \cdot f_R(r) \cdot f_{X_1}(z/r) dr \quad (\text{Since } R \perp\!\!\!\perp X_1) \\ &= \int_{|z|}^{\infty} \frac{1}{r} \cdot r \cdot \exp\left(-\frac{r^2}{2}\right) \cdot \frac{1}{\pi} \cdot \frac{r}{\sqrt{r^2 - z^2}} dr \\ &= \int_{|z|}^{\infty} \frac{r \cdot \exp\left(-\frac{r^2}{2}\right)}{\pi \sqrt{r^2 - z^2}} dr \\ &= \int_{|z|}^{\infty} \frac{\exp\left(-\frac{r^2 - z^2}{2}\right)}{2\pi} \frac{\exp\left(-\frac{z^2}{2}\right)}{\sqrt{r^2 - z^2}} d(r^2 - z^2) \end{aligned}$$

Let $t^2 = r^2 - z^2$, then

$$\begin{aligned} &= \frac{\exp\left(-\frac{z^2}{2}\right)}{2\pi} \int_0^{\infty} \exp\left(-\frac{t^2}{2}\right) \cdot \frac{1}{t} dt^2 \\ &= \frac{\exp\left(-\frac{z^2}{2}\right)}{\pi} \int_0^{\infty} \exp\left(-\frac{t^2}{2}\right) dt \end{aligned}$$

We know that

$$\int_0^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) dt = \frac{1}{2} \quad \text{Due to the property of normal distribution.}$$

Therefore

$$f_{Z_2}(z) = f_{Z_1}(z) = \frac{\sqrt{2\pi}}{2\pi} \exp\left(-\frac{z^2}{2}\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right)$$

This is exactly the probability density function of the standard normal distribution. Thus

$$Z_1 \sim \mathcal{N}(0, 1) \quad \text{and} \quad Z_2 \sim \mathcal{N}(0, 1) \quad \square$$

ECE 685 Homework 1

Xinyuan Lyu

September 22, 2024

Problem 1

Let us define $a_1 = W_1^\top X + b_1$, and $a_2 = W_2^\top h_1 + b_2$; Suppose $X \in \mathbb{R}^{m \times 1}$, $h_1 \in \mathbb{R}^{d_1 \times 1}$, $h_2 \in \mathbb{R}^{d_2 \times 1}$, that is, they are m, d_1, d_2 dims column vectors

Then

$$W_1 \in \mathbb{R}^{m \times d_1}, \quad b_1 \in \mathbb{R}^{d_1 \times 1}$$

$$W_2 \in \mathbb{R}^{d_1 \times d_2}, \quad b_2 \in \mathbb{R}^{d_2 \times 1}$$

$$W_3 \in \mathbb{R}^{d_2 \times 1}, \quad b_3 \in \mathbb{R}$$

Let $l_i = (y_i - \hat{y}_i)^2$, so for convenient, let $l = (y - \hat{y})^2$

Notice: Since all the gradient matrices below are written in Hessian form rather than Jacobian form, the chain rule should be applied from back to front.

For the b_1 and W_1 :

$$\frac{\partial l}{\partial W_3} = \frac{\partial \hat{y}}{\partial W_3} \cdot \frac{\partial l}{\partial \hat{y}} = h_2 \cdot 2(\hat{y} - y) = 2(\hat{y} - y)h_2$$

$$\frac{\partial l}{\partial b_3} = \frac{\partial \hat{y}}{\partial b_3} \cdot \frac{\partial l}{\partial \hat{y}} = 1 \cdot 2(\hat{y} - y) = 2(\hat{y} - y)$$

Then for b_2 and W_2 :

$$\frac{\partial l}{\partial b_2} = \frac{\partial a_2}{\partial b_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial l}{\partial \hat{y}}$$

$$\frac{\partial a_2}{\partial b_2} = \frac{\partial b_2}{\partial b_2} = \mathbf{I}_{d_2 \times d_2}; \quad \frac{\partial h_2}{\partial a_2} = \text{diag}\{g'(a_2)_1, \dots, g'(a_2)_{d_2}\}$$

$$\frac{\partial \hat{y}}{\partial h_2} = (W_3^\top)^\top = W_3$$

$$\begin{aligned}\frac{\partial l}{\partial b_2} &= \mathbf{I}_{d_2 \times d_2} \cdot \text{diag}\{g'(a_2)_1, \dots, g'(a_2)_{d_2}\} \cdot W_3 \cdot 2(\hat{y} - y) \\ &= 2(\hat{y} - y)(g'(a_2) \odot W_3)\end{aligned}$$

For W_2 we need more process since it is hard to derive the gradient of matrix for vector. To avoid introducing high dim tensor, we write the W_2 as following: Let

$$W_2 = \begin{pmatrix} W_{21}^\top \\ \vdots \\ W_{2d_1}^\top \end{pmatrix} \text{ where } W_{2i} = \begin{pmatrix} w_{i1} \\ \vdots \\ w_{id_2} \end{pmatrix}$$

So

$$\frac{\partial l}{\partial W_{2i}} = \frac{\partial a_2}{\partial W_{2i}} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial l}{\partial \hat{y}}$$

And we have

$$\frac{\partial a_2}{\partial W_{2i}} = \frac{\partial(h_{11}W_{21} + \dots + h_{1d_1}W_{2d_1} + b_2)}{\partial W_{2i}} = h_{1i} \cdot \mathbf{I}_{d_2 \times d_2}$$

So

$$\begin{aligned}\frac{\partial l}{\partial W_{2i}} &= h_{1i} \cdot \mathbf{I}_{d_2 \times d_2} \cdot \text{diag}\{g'(a_2)_1, \dots, g'(a_2)_{d_2}\} \cdot W_3 \cdot 2(\hat{y} - y) \\ &= 2(\hat{y} - y) \cdot h_{1i} \cdot (g'(a_2) \odot W_3)\end{aligned}$$

So combine the rows together, we have:

$$\begin{aligned}\frac{\partial l}{\partial W_2} &= 2(\hat{y} - y) \begin{bmatrix} h_{11}(g'(a_2) \odot W_3)^\top \\ \vdots \\ h_{1d_1}(g'(a_2) \odot W_3)^\top \end{bmatrix} \\ &= 2(\hat{y} - y) \cdot h_1 \cdot (g'(a_2) \odot W_3)^\top\end{aligned}$$

Then for b_1 and W_1 :

$$\frac{\partial l}{\partial b_1} = \frac{\partial a_1}{\partial b_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial l}{\partial \hat{y}}$$

Similarly as for b_2 and W_2 , we have:

$$\begin{aligned}\frac{\partial a_1}{\partial b_1} &= \mathbf{I}_{d_1 \times d_1}, \quad \frac{\partial h_1}{\partial a_1} = \text{diag}\{g'(a_1)_1, \dots, g'(a_1)_{d_1}\} \\ \frac{\partial a_2}{\partial h_1} &= (W_2^\top)^\top = W_2; \quad \frac{\partial h_2}{\partial a_2} = \text{diag}\{g'(a_2)_1, \dots, g'(a_2)_{d_2}\}; \quad \frac{\partial \hat{y}}{\partial h_2} = W_3\end{aligned}$$

So we have:

$$\begin{aligned}\frac{\partial l}{\partial b_1} &= \mathbf{I}_{d_1 \times d_1} \cdot \text{diag}\{g'(a_1)_1, \dots, g'(a_1)_{d_1}\} \cdot W_2 \cdot \text{diag}\{g'(a_2)_1, \dots, g'(a_2)_{d_2}\} \cdot W_3 \cdot 2(\hat{y} - y) \\ &= 2(\hat{y} - y)(g'(a_1) \odot [W_2(g'(a_2) \odot W_3)])\end{aligned}$$

Similarly, Let

$$W_1 = \begin{pmatrix} W_{11}^\top \\ \vdots \\ W_{1d_1}^\top \end{pmatrix} \text{ where } W_{1i} = \begin{pmatrix} w_{i1} \\ \vdots \\ w_{im} \end{pmatrix}$$

So we have:

$$\begin{aligned}\frac{\partial l}{\partial W_{1i}} &= \frac{\partial a_1}{\partial W_{1i}} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial l}{\partial \hat{y}} \\ &= x_i \cdot \mathbf{I}_{d_1 \times d_1} \cdot \text{diag}\{g'(a_1), \dots, g'(a_{1d_1})\} \cdot W_2 \cdot \text{diag}\{g'(a_2), \dots, g'(a_{2d_2})\} \cdot W_3 \cdot 2(\hat{y} - y) \\ &= 2(\hat{y} - y) \cdot x_i \cdot \{g'(a_1) \odot [W_2(g'(a_2) \odot W_3)]\}\end{aligned}$$

So combine together, we have:

$$\begin{aligned}\frac{\partial l}{\partial W_1} &= 2(\hat{y} - y) \begin{bmatrix} x_1 \cdot \{g'(a_1) \odot [W_2(g'(a_2) \odot W_3)]\}^\top \\ \vdots \\ x_m \cdot \{g'(a_1) \odot [W_2(g'(a_2) \odot W_3)]\}^\top \end{bmatrix} \\ &= 2(\hat{y} - y)x\{g'(a_1) \odot [W_2(g'(a_2) \odot W_3)]\}^\top\end{aligned}$$

So finally we have the following result. And to avoid confusion, we consistently place the sample index i on the outermost level.

$$\frac{\partial L}{\partial b_3} = \frac{1}{N} \sum_{i=1}^N \{2(\hat{y} - y)\}_i, \quad \frac{\partial L}{\partial W_3} = \frac{1}{N} \sum_{i=1}^N \{2(\hat{y} - y)h_2\}_i$$

$$\frac{\partial L}{\partial b_2} = \frac{1}{N} \sum_{i=1}^N \{2(\hat{y} - y) \cdot (g'(a_2) \odot W_3)\}_i$$

$$\frac{\partial L}{\partial W_2} = \frac{1}{N} \sum_{i=1}^N \{2(\hat{y} - y) \cdot h_1 \cdot (g'(a_2) \odot W_3)^\top\}_i$$

$$\frac{\partial L}{\partial b_1} = \frac{1}{N} \sum_{i=1}^N \left\{ 2(\hat{y} - y) \cdot \{g'(a_1) \odot [W_2 \cdot (g'(a_2) \odot W_3)]\} \right\}_i$$

$$\frac{\partial L}{\partial W_1} = \frac{1}{N} \sum_{i=1}^N \left\{ 2(\hat{y} - y) \cdot x \cdot \{g'(a_1) \odot [W_2(g'(a_2) \odot W_3)]\}^\top \right\}_i$$

Problem 2

Part 1

```
In [1]: import torch
import torch.nn as nn
import torch.optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Define the MNISTdata class.

```
In [2]: class MNISTdata(torch.utils.data.Dataset):

    def __init__(self, root, transform=None):
        self.data = pd.read_csv(root) #get the raw data
        self.transform = transform
        self.X = self.data.iloc[:, 1: ].values.astype(np.uint8) #split the data to x and y
        self.Y = self.data.iloc[:, 0].values

    def __getitem__(self, index):
        data = self.X[index].reshape(28, 28) #reshape the data to 28, 28
        label = self.Y[index]

        if self.transform:
            data = self.transform(data)

        return data, label

    def __len__(self):
        return len(self.X)
```

Read and load the data.

```
In [3]: trainset=MNISTdata("C:/Users/14857/Desktop/ECE685/homework/HW2/code/archive/mnist_train.csv",transform = tra
testset=MNISTdata("C:/Users/14857/Desktop/ECE685/homework/HW2/code/archive/mnist_test.csv",transform = tra
trainloader = torch.utils.data.DataLoader(trainset, batch_size=8, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=8, shuffle=False)
```

Draw the picture.

```
In [4]: dataiter = iter(trainloader)
images, labels = next(dataiter)
im = images[0:8]
fig , ax = plt.subplots(1,8)
for i in range(8):
    ax[i].imshow(im[i][0], cmap='gray')
plt.show()
print(f'Labels:{labels[0:8]}' )
```



Labels:tensor([2, 0, 8, 7, 2, 6, 5, 9])

Part 2

In [5]:

```
import os
from PIL import Image
import albumentations as A
from albumentations.pytorch import ToTensorV2
```

```
d:\anaconda\envs\d2l\lib\site-packages\albumentations\_init__.py:13: UserWarning: A new version of Albumentations is available: 1.4.16 (you have 1.4.14). Upgrade using: pip install -U albumentations. To disable automatic update checks, set the environment variable NO_ALBUMENTATIONS_UPDATE to 1.
check_for_updates()
```

Define the mapdata class.

In [6]:

```
class mapdata(torch.utils.data.Dataset):

    def __init__(self, image_root, mask_root, transform=None):
        self.image_root = image_root
        self.mask_root = mask_root
        self.transform = transform
        self.image_list = sorted(os.listdir(image_root)) # since all the pictures are in a file, so we need
        self.mask_list = sorted(os.listdir(mask_root))

    def __getitem__(self, index):

        image_name = os.path.join(self.image_root, self.image_list[index]) # get the absolute path of image
        mask_name = os.path.join(self.mask_root, self.mask_list[index])

        image = np.array(Image.open(image_name).convert("RGB")) # open the image to np arrays and all the
        mask = np.array(Image.open(mask_name).convert("RGB"))

        if self.transform:
            augment_data = self.transform(image=image, mask=mask) # do the augment for images and masks.
            image = augment_data['image']
            mask = augment_data['mask']

        return image, mask

    def __len__(self):
        return len(self.image_list)
```

Define the tranform, which will augment the pictures.

In [7]:

```
transform = A.Compose([
    A.RandomResizedCrop(height=256, width=256, scale=(0.8, 1.0)),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.Affine(scale=(0.5, 1.5), translate_percent=(0.3, 0.3), rotate=(-20, 20), shear=(-30, 30), p=0.5),
    A.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),
    A.pytorch.ToTensorV2()
])
```

Read and Load the data.

In [8]:

```
the_data_set = mapdata('C:/Users/14857/Desktop/ECE685/homework/HW2/code/archive/images', 'C:/Users/14857/D
the_data_loader = torch.utils.data.DataLoader(the_data_set, batch_size=4, shuffle=True)

image, mask = the_data_set[10]
print(image.shape)
print(mask.shape)

torch.Size([3, 256, 256])
torch.Size([256, 256, 3])
```

Draw the Pictures.

In [9]:

```
dataiter = iter(the_data_loader)

for turn in range(5):
```

```

images, masks = next(dataiter)
fig, ax = plt.subplots(2, 4, figsize=(15, 5))

for i in range(4):

    ax[0, i].imshow(images[i].permute(1, 2, 0))
    ax[0, i].set_title(f"image {i+1}")

    ax[1, i].imshow(masks[i])
    ax[1, i].set_title(f"mask {i+1}")

plt.tight_layout()
plt.show()

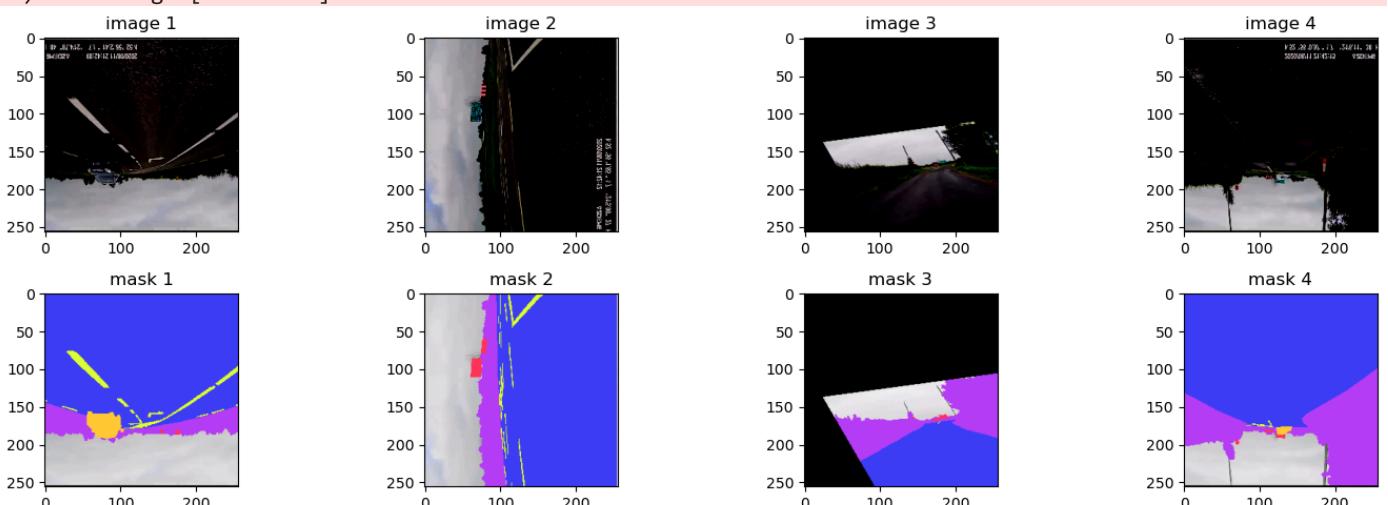
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.9058824].

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].

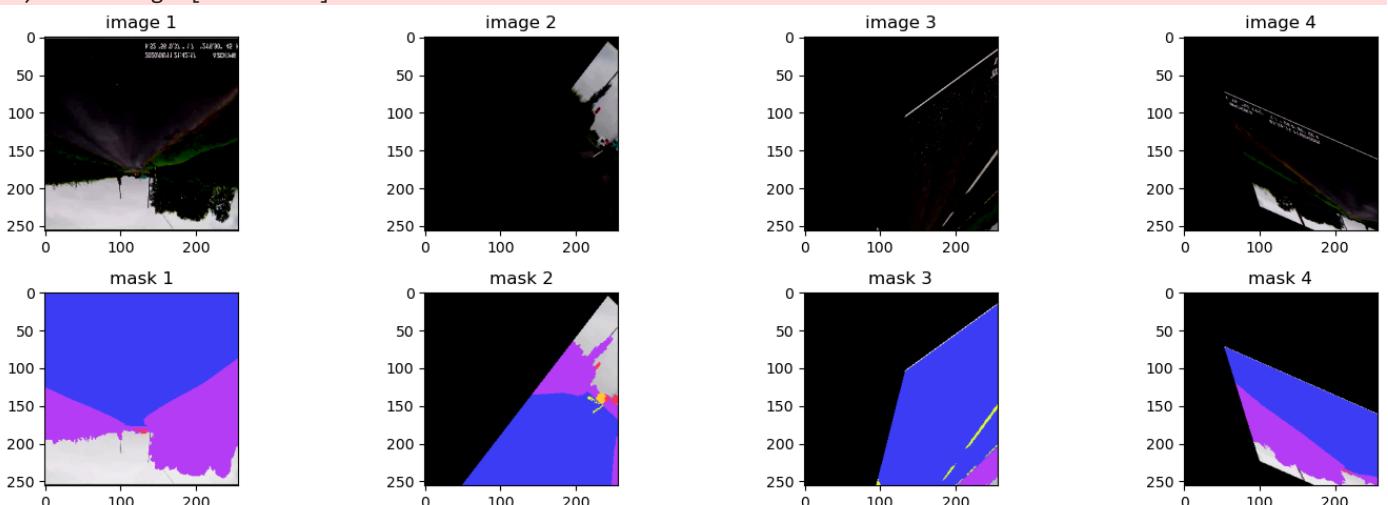


Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].

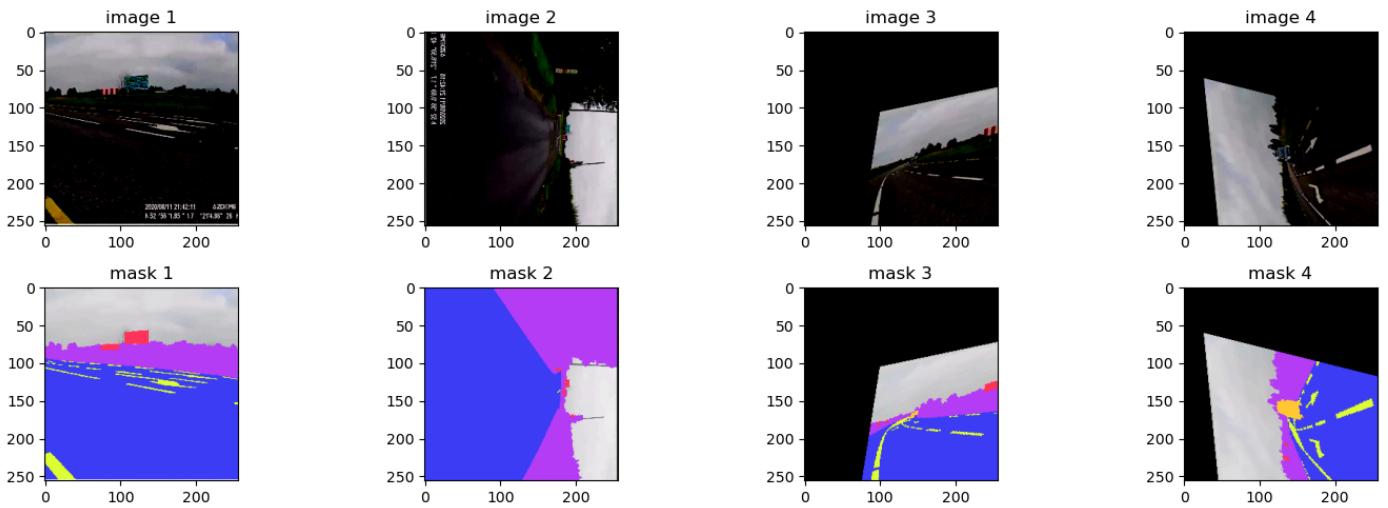
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.8117648].

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].

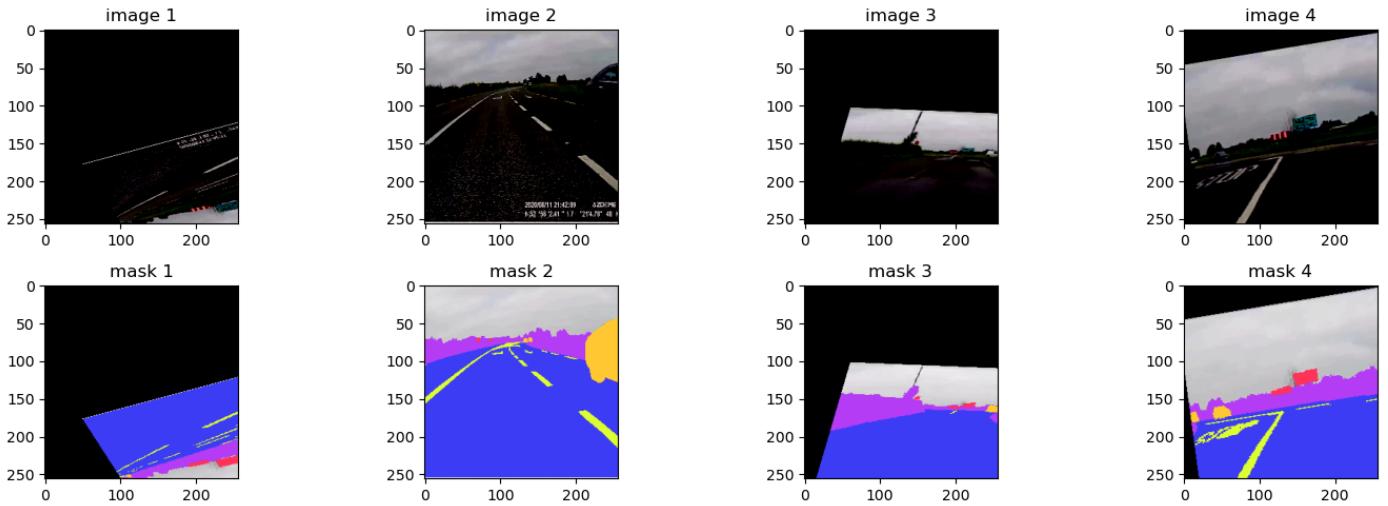
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].



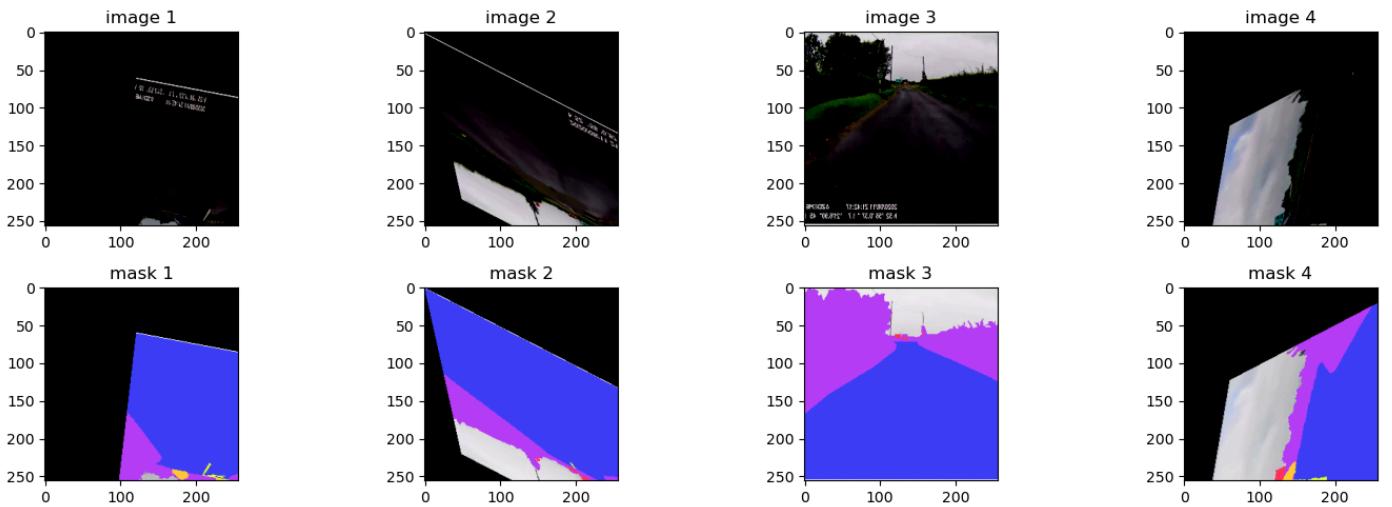
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.93725497].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.78823537].



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.91372555].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.9686275].



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.0..0.8352942].



In [10]: `%reset -f`

Problem 3

```
In [11]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.autograd as autograd

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Prepare the dataset.

```
In [12]: root_data = load_breast_cancer()
X_root = root_data.data
y_root = root_data.target

X_train_root, X_test_root, y_train_root, y_test_root = train_test_split(X_root, y_root, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_root = scaler.fit_transform(X_train_root)
X_test_root = scaler.fit_transform(X_test_root)

X_train_root = np.hstack([np.ones((X_train_root.shape[0], 1)), X_train_root]) # add the constant term.
X_test_root = np.hstack([np.ones((X_test_root.shape[0], 1)), X_test_root]) # add the constant term.

X_train_root = torch.from_numpy(X_train_root).float()
y_train_root = torch.from_numpy(y_train_root).float()
X_test_root = torch.from_numpy(X_test_root).float()
y_test_root = torch.from_numpy(y_test_root).float()

print(X_train_root.shape)
print(X_train_root.shape)
print(X_train_root[0])

torch.Size([398, 31])
torch.Size([398, 31])
tensor([ 1.0000, -0.5206, -0.3630, -0.4655, -0.5522,  0.6360,  0.4767, -0.1807,
        -0.5793,  0.0030,  1.1679, -0.8164,  0.4358, -0.3619, -0.5394,  0.8984,
        0.3856,  0.6826, -0.0638,  0.1989,  0.8297, -0.7349, -0.4486, -0.5643,
       -0.6622,  0.5583,  0.0179, -0.1864, -0.6758, -0.5811,  0.5192])
```

Model Specification.

```
In [13]: # define the Logistic regression function.
def logistic_reg(X, W):
    a = torch.matmul(X, W)
    a = torch.clamp(a, min=-88, max=88)
    return 1 / (1 + torch.exp(-(a)))

# define the bce loss function.
def BCELoss(W, X, y):
    a = torch.matmul(X, W)
    a = torch.clamp(a, min=-88, max=88) # avoid |a| is too large that the exp will fail.
    prediction = 1 / (1 + torch.exp(-(a)))
    prediction = torch.clamp(prediction, min=1e-4, max=1 - 1e-4) # avoid prediction is too close to 0 or 1
    average_loss = -torch.mean(y * torch.log(prediction) + (1 - y) * torch.log(1 - prediction))
    return average_loss

# define the auto-computing gradient function.
def gradient_function(W, X, y):
    return autograd.grad(BCELoss(W, X, y), W, create_graph=True)[0]

# define the computing hessian matrix function.
def hessian_W(W, X, y):
    grad = gradient_function(W, X, y)
    n = W.shape[0]
    H = torch.zeros((n, n))

    for i in range(n):
        grad2 = torch.autograd.grad(grad[i], W, create_graph=True)[0]
        H[i] = grad2.view(-1)

    return H

# define the computing diagonal approximate hessian matrix function.
def approximate_hessian_W(W, X, y):
    grad = gradient_function(W, X, y)
    n = W.shape[0]
    H = torch.zeros((n, n))

    for i in range(n):
        grad2 = torch.autograd.grad(grad[i], W, create_graph=True)[0]
        H[i][i] = grad2.view(-1)[i].item()

    return H
```

Train the Models.

Exact expression for the Hessian

```
In [14]: learn_rate = 0.1 # set the learning rate.
num_epochs = 50 # set the number of epochs.
num_inputs=31 # the feature is 30+1 which contains the constant term.

# initialize the W, since this is logistic regression so for sigmoid function, W=0 is suitable.
W = torch.zeros(num_inputs, requires_grad=True)
print(W.shape)

# Prepare for the plotting later.
train_losses_1 = []
test_losses_1 = []
train_accuracies_1 = []
test_accuracies_1 = []

torch.Size([31])
```

```
In [15]: for epoch in range(num_epochs):

    correct_train = 0
    total_train = 0

    # get the data.
    X = X_train_root
```

```

y = y_train_root

# compute the prediction and Loss in this turn; and get the accuracy.
y_hat = logistic_reg(X, W)
y_hat = torch.round(y_hat)
ls_train = BCELoss(W, X, y)

correct_train += (y_hat == y).sum().item()
total_train = y.size(0)

train_losses_1.append(ls_train.item())
train_accuracy = correct_train / total_train
train_accuracies_1.append(train_accuracy)

# do the same computing for the prediction and Loss in this turn; and get the accuracy, in the test do
with torch.no_grad():

    total_test_loss = 0
    correct_test = 0
    total_test = 0

    X_test = X_test_root
    y_test = y_test_root

    y_hat_test = logistic_reg(X_test,W)
    y_hat_test = torch.round(y_hat_test)
    test_loss = BCELoss(W, X, y)

    correct_test += (y_hat_test == y_test).sum().item()
    total_test += y_test.size(0)

    test_losses_1.append(test_loss.item())
    test_accuracy = correct_test / total_test
    test_accuracies_1.append(test_accuracy)

#Compute the gradient, Hessian and inverse of Hessian.
Gradient=gradient_function(W, X, y)
Hessian=hessian_W(W, X, y)
Hessian_inverse = torch.inverse(Hessian)
#Save the data
previous_W = W.clone()
previous_Gradient = Gradient.clone()
previous_Hessian = Hessian.clone()
previous_Hessian_inverse = Hessian_inverse.clone()

# Renew the Weight.
W = W - learn_rate*torch.matmul(Hessian_inverse, Gradient)

if torch.all(W == 0) or torch.isnan(W).any():
    print("Error: W is all 0 or NaN")
    print("Previous W:", previous_W)
    print("Previous Gradient:", previous_Gradient)
    print("Previous Hessian:", previous_Hessian)
    print("Previous Hessian_inverse:", previous_Hessian_inverse)
    break

```

Diagonal approximation for Hessian

```

In [16]: learn_rate = 0.1 # set the learning rate.
num_epochs = 50
num_inputs=31

W = torch.zeros(num_inputs, requires_grad=True)
print(W.shape)

train_losses_2 = [] # Prepare for the plotting later.
test_losses_2 = []
train_accuracies_2 = []
test_accuracies_2 = []

```

```
torch.Size([31])
```

```
In [17]: for epoch in range(num_epochs):

    correct_train = 0
    total_train = 0

    # get the data.
    X = X_train_root
    y = y_train_root

    # compute the prediction and Loss in this turn; and get the accuracy.
    y_hat = logistic_reg(X, W)
    y_hat = torch.round(y_hat)
    ls_train = BCELoss(W, X, y)

    correct_train += (y_hat == y).sum().item()
    total_train = y.size(0)

    train_losses_2.append(ls_train.item())
    train_accuracy = correct_train / total_train
    train_accuracies_2.append(train_accuracy)

# do the same computing for the prediction and Loss in this turn; and get the accuracy, in the test do
with torch.no_grad():

    total_test_loss = 0
    correct_test = 0
    total_test = 0

    X_test = X_test_root
    y_test = y_test_root

    y_hat_test = logistic_reg(X_test,W)
    y_hat_test = torch.round(y_hat_test)
    test_loss = BCELoss(W, X, y)

    correct_test += (y_hat_test == y_test).sum().item()
    total_test += y_test.size(0)

    test_losses_2.append(test_loss.item())
    test_accuracy = correct_test / total_test
    test_accuracies_2.append(test_accuracy)

#Compute the gradient, Hessian and inverse of Hessian.
Gradient=gradient_function(W, X, y)
Hessian=approximate_hessian_W(W, X, y)
diag = torch.diagonal(Hessian)
inv_diag = 1.0 / diag
Hessian_inverse = torch.diag(inv_diag)

# Rewnew the wights.
W = W - learn_rate*torch.matmul(Hessian_inverse, Gradient)
```

Plot and Compare

```
In [18]: def plot_the_model(ax, y1, y2, plot_type='loss'):

    if plot_type == 'loss':
        ax.plot(y1, label='Training Loss')
        ax.plot(y2, label='Test Loss')
        ax.set_ylabel('Loss')
    elif plot_type == 'accuracy':
        ax.plot(y1, label='Training Accuracy')
        ax.plot(y2, label='Test Accuracy')
        ax.set_ylabel('Accuracy')
    ax.set_xlabel('Epoch')
    ax.legend()

fig, axes = plt.subplots(2, 2, figsize=(10, 7))

plot_the_model(axes[0, 0], train_losses_1, test_losses_1, plot_type='loss')
```

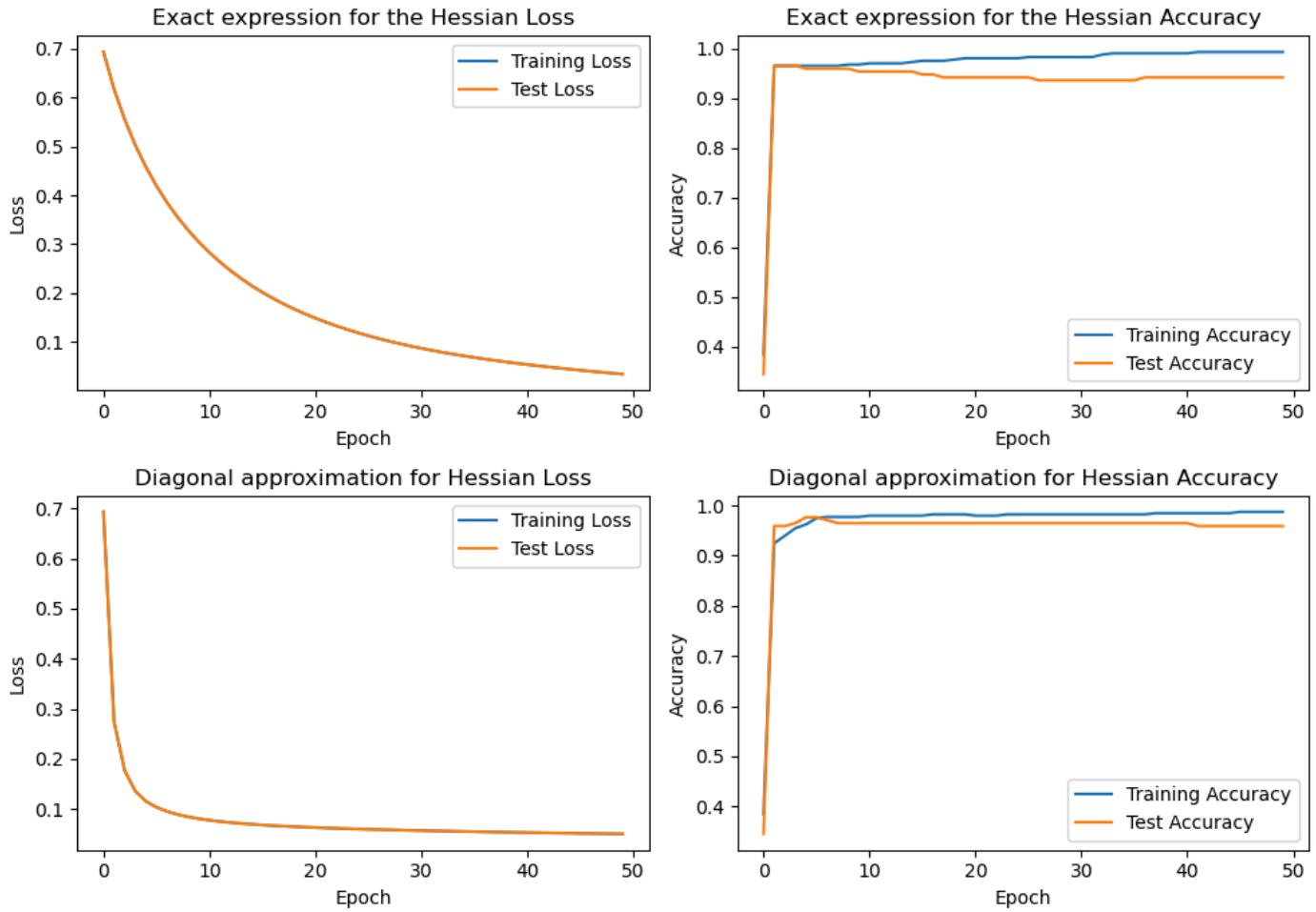
```

axes[0, 0].set_title('Exact expression for the Hessian Loss')
plot_the_model(axes[0, 1], train_accuracies_1, test_accuracies_1, plot_type='accuracy')
axes[0, 1].set_title('Exact expression for the Hessian Accuracy')

plot_the_model(axes[1, 0], train_losses_2, test_losses_2, plot_type='loss')
axes[1, 0].set_title('Diagonal approximation for Hessian Loss')
plot_the_model(axes[1, 1], train_accuracies_2, test_accuracies_2, plot_type='accuracy')
axes[1, 1].set_title('Diagonal approximation for Hessian Accuracy')

plt.tight_layout()
plt.show()

```



Problem 1: Object Detection with Convolutional Neural Networks (CNNs)

```
In [1]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
from torch.utils.data import Dataset
import torchvision.transforms as T
import matplotlib.pyplot as plt
import numpy as np
import math
import os
import pandas as pd
from PIL import Image
import albumentations as A
from albumentations.pytorch import ToTensorV2
```

```
d:\anaconda\envs\d2l\lib\site-packages\albumentations\__init__.py:13: UserWarning: A new version of Albumentations is available: 1.4.18 (you have 1.4.14). Upgrade using: pip install -U albumentations. To disable automatic update checks, set the environment variable NO_ALBUMENTATIONS_UPDATE to 1.
check_for_updates()
```

1.1 Data Preprocessing

Firstly, define the face data class.

```
In [2]: class facedata(Dataset):
    def __init__(self, image_root, box_root, transform=None):
        self.boxes = pd.read_csv(box_root) # The table of the box data.
        self.image_root = image_root
        self.transform = transform

    def __getitem__(self, index):
        image_name = os.path.join(self.image_root, self.boxes.iloc[index, 0])
        image = Image.open(image_name).convert("RGB")

        x_min = self.boxes.iloc[index, 3]
        y_min = self.boxes.iloc[index, 4]
        x_max = self.boxes.iloc[index, 5]
        y_max = self.boxes.iloc[index, 6]
        box = [x_min, y_min, x_max, y_max]

        if self.transform:
            image = np.array(image)
            transformed = self.transform(image=image, bboxes=[box])
            image = transformed['image']
            box = transformed['bboxes'][0]

        return image, torch.tensor(box, dtype=torch.float32)

    def __len__(self):
        return len(self.boxes)
```

Secondly, define the spatial transformations.

```
In [3]: transformations = A.Compose([
    A.Resize(224, 224),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.ElasticTransform(alpha=120, sigma=120 * 0.05, alpha_affine=None, p=0.5),
    A.pytorch.ToTensorV2()
], bbox_params=A.BboxParams(format='pascal_voc', label_fields=[]))

# For test data, we do not need augmentation but should resize it for later model.
test_transformations = A.Compose([
    A.Resize(224, 224),
    A.pytorch.ToTensorV2()
], bbox_params=A.BboxParams(format='pascal_voc', label_fields=[]))
```

Thirdly, divide the data into train and test, and drop the duplicates since our model just detect single target.

```
In [4]: from sklearn.model_selection import train_test_split

boxes = pd.read_csv('C:/Users/14857/Desktop/ECE685/homework/HW3/archive/faces.csv')
print(len(boxes))
boxes = boxes.groupby('image_name').apply(lambda x: x.sample(1)).reset_index(drop=True)
print(len(boxes))
```

```

train_boxes, test_boxes = train_test_split(boxes, test_size=0.2, random_state=10)
train_boxes.to_csv('C:/Users/14857/Desktop/ECE685/homework/HW3/archive/train_boxes.csv', index=False)
test_boxes.to_csv('C:/Users/14857/Desktop/ECE685/homework/HW3/archive/test_boxes.csv', index=False)

train_dataset = facedata('C:/Users/14857/Desktop/ECE685/homework/HW3/archive/images', 'C:/Users/14857/Desktop/ECE685/homework/HW3/archive/train')
test_dataset = facedata('C:/Users/14857/Desktop/ECE685/homework/HW3/archive/images', 'C:/Users/14857/Desktop/ECE685/homework/HW3/archive/test')

```

3350

2204

```
C:\Users\14857\AppData\Local\Temp\ipykernel_22864\2344910187.py:5: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping column s. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groupwise=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
boxes = boxes.groupby('image_name').apply(lambda x: x.sample(1)).reset_index(drop=True)
```

Fourthly, draw the pictures.

```
In [5]: def draw_sample(dataset, indices):

    fig, axes = plt.subplots(1, len(indices), figsize=(16, 4))

    for i, idx in enumerate(indices):
        sample = dataset[idx]
        image = sample[0].permute(1, 2, 0)
        bbox = sample[1]

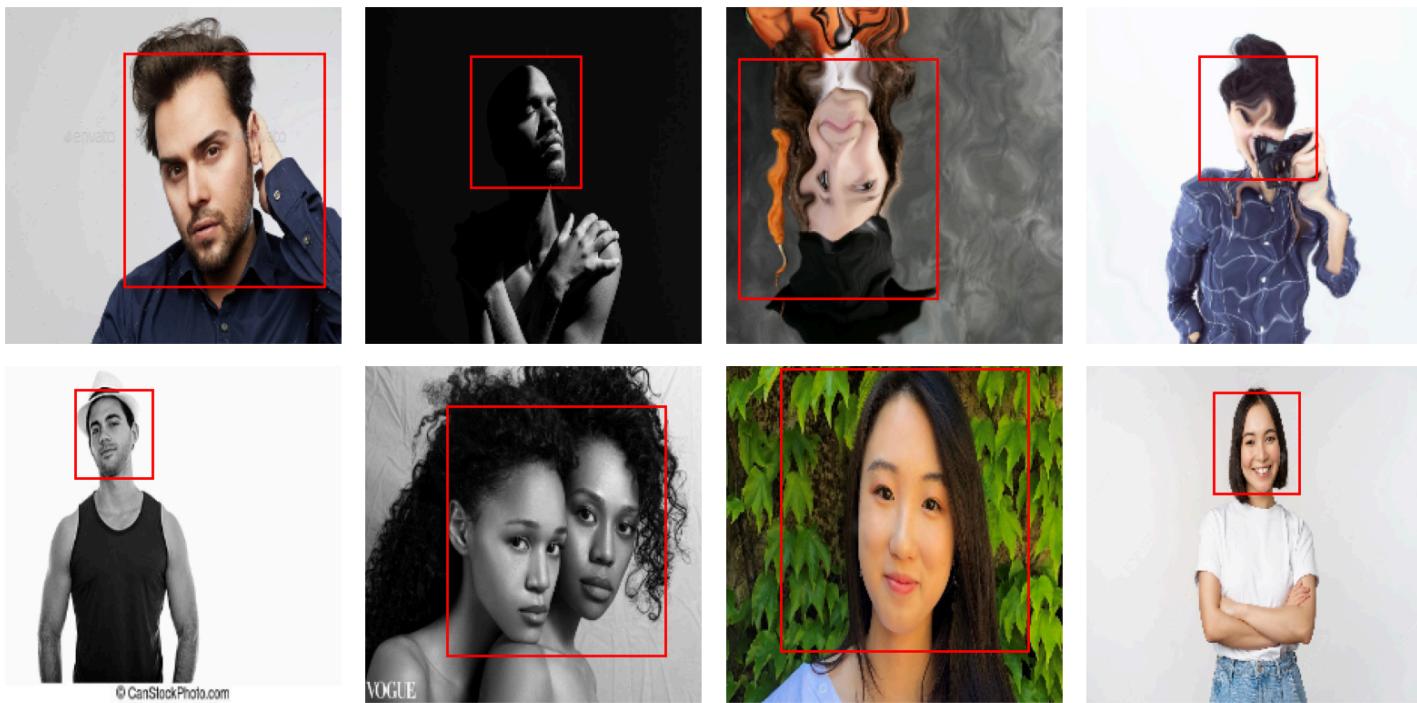
        axes[i].imshow(image)

        x_min, y_min, x_max, y_max = bbox
        rect = plt.Rectangle((x_min, y_min), x_max - x_min, y_max - y_min,
                             linewidth=2, edgecolor='r', facecolor='none')
        axes[i].add_patch(rect)

        axes[i].axis('off')

    plt.tight_layout()
    plt.show()

draw_sample(train_dataset, indices=[0, 1, 2, 3]) # The train data sets.
draw_sample(test_dataset, indices=[0, 1, 2, 3]) # The test data sets.
```



1.2 Object Detection with Pre-trained Feature Extractor

```
In [6]: import torchvision.models as models
from torchvision.ops import generalized_box_iou_loss
from torchvision.ops import box_iou as IoU
```

Firstly, Define the model based on backbone, we use resnet50 as backbone here.

```
In [7]: class BoundingBoxRegressor(nn.Module):
    def __init__(self, freeze_backbone=True):
        super(BoundingBoxRegressor, self).__init__()

        self.normalize = T.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))

        self.feature_extractor = models.resnet50(weights='IMAGENET1K_V1').to(device)
        in_features = self.feature_extractor.fc.in_features
        self.feature_extractor.fc = nn.Identity()

    if freeze_backbone:
        for param in self.feature_extractor.parameters():
```

```

        param.requires_grad = False
    else:
        for param in self.feature_extractor.parameters():
            param.requires_grad = True

    self.regressor = nn.Sequential(
        nn.Linear(in_features, 512),
        nn.Linear(512, 206),
        nn.Linear(206, 103),
        nn.Linear(103, 4)
    )

    for param in self.regressor.parameters():
        param.requires_grad = True

    def forward(self, x):
        x = self.normalize(x)
        features = self.feature_extractor(x)
        bbox_pred = self.regressor(features)

    return bbox_pred

```

Secondly, Define the train process. (Rewrite from the trainer present in discussion section)

```

In [8]: def train(model, device, train_loader, batch_size, criterion, optimizer, epoch, train_losses, train_IoUs):
    model.train()

    train_loss = 0
    train_IoU = 0
    batch_index = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        batch_index = batch_idx
        data, target = data.to(device), target.to(device)
        data, target = data.float(), target.float()

        if len(data) != batch_size:
            continue

        optimizer.zero_grad()

        output = model(data)
        loss = criterion(output, target)

        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        for i in range(batch_size):
            pred_box = output[i]
            true_box = target[i]
            pred_box = output[i].unsqueeze(0)
            true_box = target[i].unsqueeze(0)
            iou = IoU(pred_box, true_box).item()
            train_IoU += iou

        if batch_idx % (len(train_loader)//2) == 0:
            print('Train({}): Loss: {:.4f}, IoU: {:.4f}'.format(
                epoch, 100. * batch_idx / len(train_loader), train_loss/(batch_idx+1), 100. * train_IoU/(batch_size*(batch_idx+1)))))

        train_losses.append(train_loss/(batch_index+1))
        train_IoUs.append(train_IoU/(batch_size*(batch_index+1)))

def test(model, device, test_loader, batch_size, criterion, epoch, test_losses, test_IoUs):
    model.eval()
    test_loss = 0
    test_IoU = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            data, target = data.float(), target.float()

            if len(data) != batch_size:
                continue

            output = model(data)

            test_loss += criterion(output, target).item()
            for i in range(batch_size):
                pred_box = output[i]
                true_box = target[i]
                pred_box = output[i].unsqueeze(0)
                true_box = target[i].unsqueeze(0)
                iou = IoU(pred_box, true_box).item()
                test_IoU += iou

        test_loss = (test_loss*batch_size) / len(test_loader.dataset)
        test_IoU = test_IoU / len(test_loader.dataset)
        test_losses.append(test_loss)
        test_IoUs.append(test_IoU)

```

```
print('Test({}): Loss: {:.4f}, IoU: {:.4f}%'.format(
```

```
epoch, test_loss, 100. * test_IoU))
```

Thirdly, Train the freezed model.

```
In [9]: device = 'cuda'  
device = torch.device(device)
```

```
In [10]: model_freeze = BoundingBoxRegressor().to(device)  
criterion = nn.SmoothL1Loss()  
trained_params = filter(lambda p: p.requires_grad, model_freeze.parameters())  
optimizer = torch.optim.Adam(trained_params, lr=0.0001)  
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
```

```
In [11]: train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)  
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
In [12]: num_epochs = 20  
batch_size = 32  
train_losses_freeze = []  
test_losses_freeze = []  
train_IoUs_freeze = []  
test_IoUs_freeze = []  
  
for epoch in range(1, num_epochs + 1):  
    train(model_freeze, device, train_loader, batch_size, criterion, optimizer, epoch, train_losses_freeze, train_IoUs_freeze)  
    test(model_freeze, device, test_loader, batch_size, criterion, epoch, test_losses_freeze, test_IoUs_freeze)
```

Train(1)[0%]: Loss: 107.1134, IoU: 0.0000%
Train(1)[50%]: Loss: 105.7426, IoU: 0.0000%
Test(1): Loss: 37.4105, IoU: 4.6857%
Train(2)[0%]: Loss: 44.5937, IoU: 2.5791%
Train(2)[50%]: Loss: 37.9039, IoU: 13.4521%
Test(2): Loss: 33.4721, IoU: 13.7142%
Train(3)[0%]: Loss: 35.0977, IoU: 13.5766%
Train(3)[50%]: Loss: 36.9243, IoU: 13.8204%
Test(3): Loss: 33.4420, IoU: 13.2076%
Train(4)[0%]: Loss: 30.6216, IoU: 20.0096%
Train(4)[50%]: Loss: 35.9673, IoU: 15.1425%
Test(4): Loss: 32.4320, IoU: 14.5082%
Train(5)[0%]: Loss: 37.3387, IoU: 11.9081%
Train(5)[50%]: Loss: 36.6842, IoU: 14.6729%
Test(5): Loss: 32.9540, IoU: 13.1431%
Train(6)[0%]: Loss: 37.8008, IoU: 13.6779%
Train(6)[50%]: Loss: 36.3489, IoU: 15.7000%
Test(6): Loss: 32.8169, IoU: 12.5595%
Train(7)[0%]: Loss: 40.1048, IoU: 9.3592%
Train(7)[50%]: Loss: 35.9128, IoU: 15.5333%
Test(7): Loss: 30.8164, IoU: 14.6628%
Train(8)[0%]: Loss: 35.7973, IoU: 12.8183%
Train(8)[50%]: Loss: 35.7249, IoU: 16.8459%
Test(8): Loss: 32.7556, IoU: 12.9835%
Train(9)[0%]: Loss: 36.3662, IoU: 12.8384%
Train(9)[50%]: Loss: 36.1398, IoU: 15.4729%
Test(9): Loss: 31.4469, IoU: 13.9934%
Train(10)[0%]: Loss: 31.7258, IoU: 19.3408%
Train(10)[50%]: Loss: 35.1433, IoU: 16.6951%
Test(10): Loss: 31.5513, IoU: 13.3725%
Train(11)[0%]: Loss: 35.4014, IoU: 15.1041%
Train(11)[50%]: Loss: 34.7345, IoU: 16.5593%
Test(11): Loss: 32.0198, IoU: 12.8700%
Train(12)[0%]: Loss: 32.6085, IoU: 19.8638%
Train(12)[50%]: Loss: 34.5332, IoU: 17.3144%
Test(12): Loss: 31.1530, IoU: 14.4616%
Train(13)[0%]: Loss: 36.3003, IoU: 18.1582%
Train(13)[50%]: Loss: 34.8242, IoU: 17.0479%
Test(13): Loss: 29.5593, IoU: 16.3307%
Train(14)[0%]: Loss: 31.6988, IoU: 18.7994%
Train(14)[50%]: Loss: 34.6328, IoU: 17.2020%
Test(14): Loss: 28.7458, IoU: 17.4596%
Train(15)[0%]: Loss: 33.3648, IoU: 18.0659%
Train(15)[50%]: Loss: 33.4994, IoU: 18.5468%
Test(15): Loss: 29.1038, IoU: 17.9824%
Train(16)[0%]: Loss: 35.5309, IoU: 16.5881%
Train(16)[50%]: Loss: 33.6779, IoU: 18.0851%
Test(16): Loss: 26.9904, IoU: 20.4777%
Train(17)[0%]: Loss: 33.9436, IoU: 15.1996%
Train(17)[50%]: Loss: 32.4749, IoU: 18.9965%
Test(17): Loss: 27.1103, IoU: 22.2234%
Train(18)[0%]: Loss: 33.2289, IoU: 18.6303%
Train(18)[50%]: Loss: 32.3624, IoU: 19.7700%
Test(18): Loss: 26.5488, IoU: 23.1957%
Train(19)[0%]: Loss: 34.5886, IoU: 16.4730%
Train(19)[50%]: Loss: 32.0087, IoU: 19.8285%
Test(19): Loss: 26.1528, IoU: 23.6127%
Train(20)[0%]: Loss: 27.9429, IoU: 23.4397%
Train(20)[50%]: Loss: 31.3822, IoU: 21.0365%
Test(20): Loss: 26.5507, IoU: 22.2917%

Fourthly, Evaluate the performance.

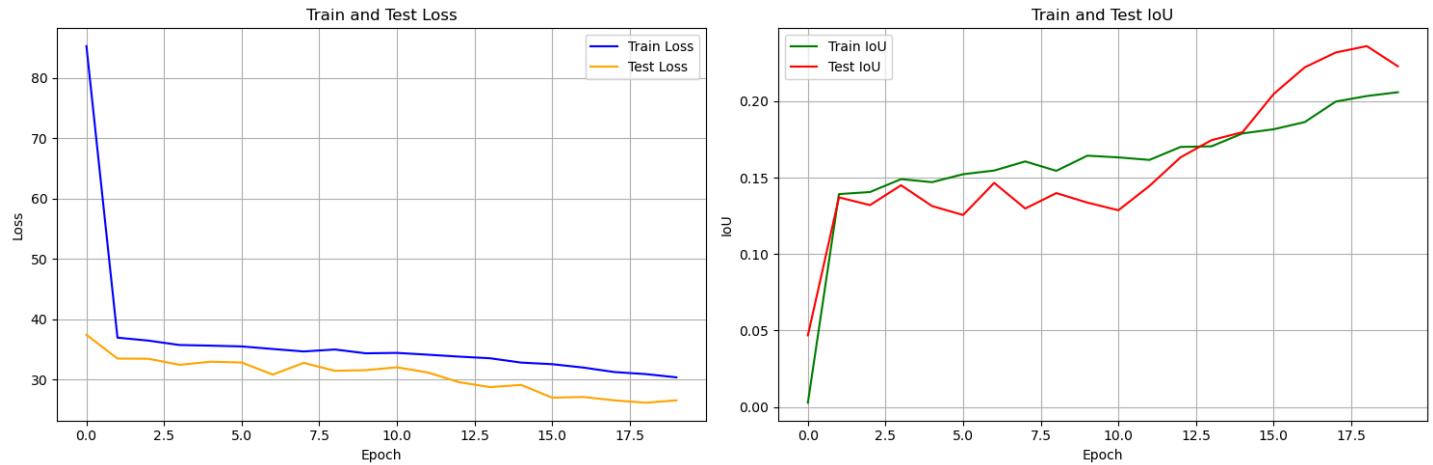
In [13]:

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].plot(train_losses_freeze, label='Train Loss', color='blue')
axes[0].plot(test_losses_freeze, label='Test Loss', color='orange')
axes[0].set_title('Train and Test Loss')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Loss')
axes[0].legend()
axes[0].grid(True)

axes[1].plot(train_IoUs_freeze, label='Train IoU', color='green')
axes[1].plot(test_IoUs_freeze, label='Test IoU', color='red')
axes[1].set_title('Train and Test IoU')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('IoU')
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.show()
```



In [14]:

```
def draw_sample_predict(dataset, model, indices, device='cuda'):
    model.eval()
    fig, axes = plt.subplots(1, len(indices), figsize=(16, 4))

    for i, idx in enumerate(indices):
        sample = dataset[idx]
        image = sample[0].unsqueeze(0).to(device)
        image = image.float()
        true_bbox = sample[1]

        with torch.no_grad():
            pred_bbox = model(image).squeeze(0).cpu().numpy()

        image_np = sample[0].permute(1, 2, 0).cpu().numpy()

        axes[i].imshow(image_np)

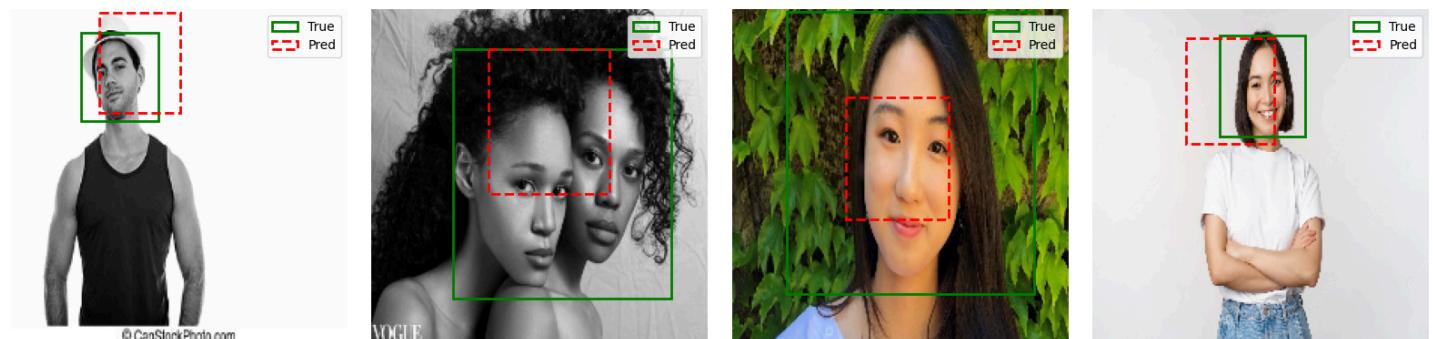
        x_min, y_min, x_max, y_max = true_bbox
        rect_true = plt.Rectangle((x_min, y_min), x_max - x_min, y_max - y_min,
                                  linewidth=2, edgecolor='g', facecolor='none', label='True')
        axes[i].add_patch(rect_true)

        pred_x_min, pred_y_min, pred_x_max, pred_y_max = pred_bbox
        rect_pred = plt.Rectangle((pred_x_min, pred_y_min), pred_x_max - pred_x_min, pred_y_max - pred_y_min,
                                  linewidth=2, edgecolor='r', facecolor='none', linestyle='--', label='Pred')
        axes[i].add_patch(rect_pred)

        axes[i].legend(loc='upper right')
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()
```

```
draw_sample_predict(test_dataset, model_freeze, indices=[0, 1, 2, 3], device='cuda')
```



1.3 Object Detection with End-to-End Fine-Tuning

We just unfreeze the backbone model and train its parameters together with regressor.

```
In [15]: model_unfreeze = BoundingBoxRegressor(freeze_backbone=False).to(device)
criterion = nn.SmoothL1Loss()
trained_params = filter(lambda p: p.requires_grad, model_unfreeze.parameters())
optimizer = torch.optim.Adam(trained_params, lr=0.0001)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

In [16]: num_epochs = 20
batch_size = 32
train_losses_unfreeze = []
test_losses_unfreeze = []
train_IoUs_unfreeze = []
test_IoUs_unfreeze = []

for epoch in range(1, num_epochs + 1):
    train(model_unfreeze, device, train_loader, batch_size, criterion, optimizer, epoch, train_losses_unfreeze, train_IoUs_unfreeze)
    test(model_unfreeze, device, test_loader, batch_size, criterion, epoch, test_losses_unfreeze, test_IoUs_unfreeze)

Train(1)[0%]: Loss: 110.0982, IoU: 0.0000%
Train(1)[50%]: Loss: 104.9901, IoU: 0.0000%
Test(1): Loss: 31.3872, IoU: 12.1240%
Train(2)[0%]: Loss: 32.9646, IoU: 15.8100%
Train(2)[50%]: Loss: 33.0113, IoU: 19.4700%
Test(2): Loss: 29.1170, IoU: 17.0668%
Train(3)[0%]: Loss: 28.2082, IoU: 25.5726%
Train(3)[50%]: Loss: 30.9625, IoU: 20.5765%
Test(3): Loss: 29.6389, IoU: 16.8309%
Train(4)[0%]: Loss: 31.5087, IoU: 20.7868%
Train(4)[50%]: Loss: 29.6242, IoU: 22.4490%
Test(4): Loss: 27.1182, IoU: 19.9335%
Train(5)[0%]: Loss: 28.1664, IoU: 22.7403%
Train(5)[50%]: Loss: 29.9899, IoU: 21.9182%
Test(5): Loss: 27.9804, IoU: 19.1233%
Train(6)[0%]: Loss: 29.1762, IoU: 22.5122%
Train(6)[50%]: Loss: 29.5591, IoU: 21.8913%
Test(6): Loss: 24.9364, IoU: 22.6841%
Train(7)[0%]: Loss: 29.2265, IoU: 19.5882%
Train(7)[50%]: Loss: 25.6985, IoU: 27.4438%
Test(7): Loss: 18.6393, IoU: 35.3309%
Train(8)[0%]: Loss: 24.8694, IoU: 28.3896%
Train(8)[50%]: Loss: 20.9469, IoU: 34.3704%
Test(8): Loss: 16.9265, IoU: 38.9307%
Train(9)[0%]: Loss: 20.1279, IoU: 34.7158%
Train(9)[50%]: Loss: 20.1605, IoU: 36.1457%
Test(9): Loss: 16.9302, IoU: 37.4639%
Train(10)[0%]: Loss: 13.9551, IoU: 43.7116%
Train(10)[50%]: Loss: 19.5373, IoU: 38.0306%
Test(10): Loss: 13.8270, IoU: 44.2648%
Train(11)[0%]: Loss: 16.2100, IoU: 45.2525%
Train(11)[50%]: Loss: 16.5322, IoU: 44.2315%
Test(11): Loss: 13.1277, IoU: 47.1524%
Train(12)[0%]: Loss: 18.8270, IoU: 38.3069%
Train(12)[50%]: Loss: 14.2585, IoU: 48.1373%
Test(12): Loss: 10.9841, IoU: 53.6854%
Train(13)[0%]: Loss: 11.5899, IoU: 51.2312%
Train(13)[50%]: Loss: 14.5221, IoU: 48.2171%
Test(13): Loss: 13.4054, IoU: 45.3727%
Train(14)[0%]: Loss: 17.0873, IoU: 39.2255%
Train(14)[50%]: Loss: 14.1399, IoU: 48.4099%
Test(14): Loss: 12.6443, IoU: 45.7443%
Train(15)[0%]: Loss: 11.9311, IoU: 51.3749%
Train(15)[50%]: Loss: 13.6309, IoU: 49.5447%
Test(15): Loss: 11.3667, IoU: 50.8058%
Train(16)[0%]: Loss: 14.9576, IoU: 49.8463%
Train(16)[50%]: Loss: 12.1904, IoU: 53.8029%
Test(16): Loss: 10.6327, IoU: 55.0364%
Train(17)[0%]: Loss: 10.6277, IoU: 56.0676%
Train(17)[50%]: Loss: 12.2277, IoU: 52.6095%
Test(17): Loss: 10.5099, IoU: 54.7021%
Train(18)[0%]: Loss: 15.0449, IoU: 49.3998%
Train(18)[50%]: Loss: 11.8952, IoU: 55.1582%
Test(18): Loss: 10.5323, IoU: 55.1385%
Train(19)[0%]: Loss: 11.6282, IoU: 53.8874%
Train(19)[50%]: Loss: 11.5273, IoU: 55.1097%
Test(19): Loss: 11.0535, IoU: 55.0947%
Train(20)[0%]: Loss: 8.6933, IoU: 59.8899%
Train(20)[50%]: Loss: 12.3064, IoU: 52.7703%
Test(20): Loss: 11.0984, IoU: 53.8732%
```

Evaluate the performance.

```
In [17]: fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].plot(train_losses_unfreeze, label='Train Loss', color='blue')
axes[0].plot(test_losses_unfreeze, label='Test Loss', color='orange')
axes[0].set_title('Train and Test Loss')
axes[0].set_xlabel('Epoch')
```

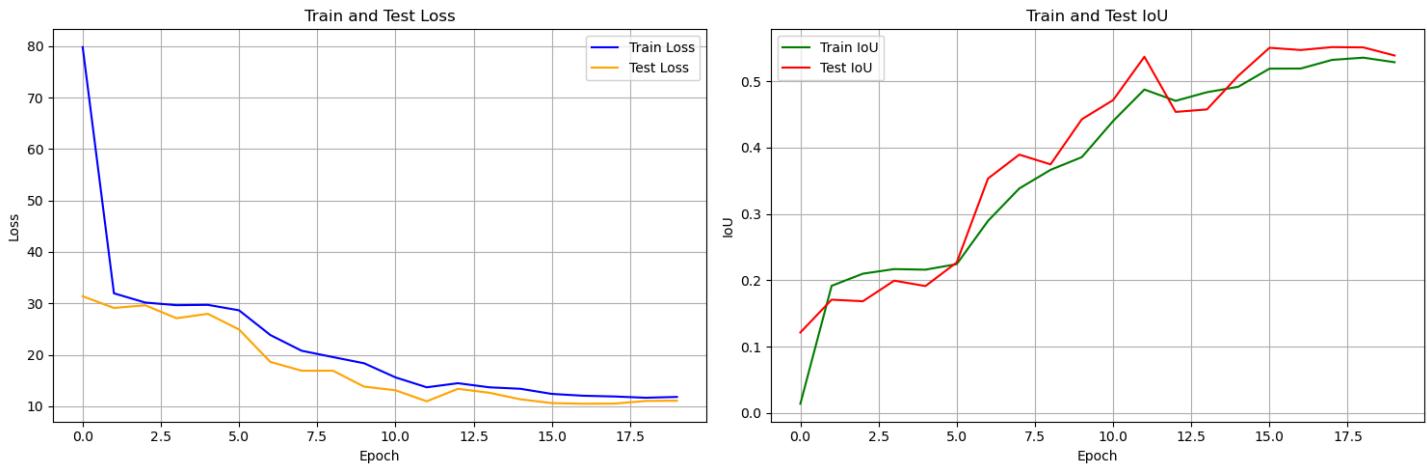
```

axes[0].set_ylabel('Loss')
axes[0].legend()
axes[0].grid(True)

axes[1].plot(train_IoUs_unfreeze, label='Train IoU', color='green')
axes[1].plot(test_IoUs_unfreeze, label='Test IoU', color='red')
axes[1].set_title('Train and Test IoU')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('IoU')
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.show()

```



```
In [18]: draw_sample_predict(test_dataset, model_unfreeze, indices=[0, 1, 2, 3], device='cuda')
```



Conclusion: It is clearly evident that training the bounding box prediction module along with the backbone significantly improves the model's performance. The model's IoU has increased remarkably from around 20% to over 50%. This indicates that fine-tuning the backbone model is crucial in transfer learning.

1.4 What if there are multiple objects?

First, my model cannot handle cases with either no objects or multiple objects. This is because my model's output is always a fixed-length vector of 4, representing the coordinates of the bottom-left and top-right corners of a single bounding box. Therefore, it always outputs only one bounding box coordinate. The issue is that the length of the output layer should be variable, not constant, but our model forcibly sets it to a constant.

Second, to address this problem, we can draw on the ideas from algorithms like R-CNN and YOLO. These algorithms first generate a large number of anchors, and then use selective search algorithms to select candidate region proposals. After that, the model performs bounding box regression on these region proposals, and finally, the Non-Maximum Suppression (NMS) algorithm is used to select the final bounding boxes. This approach works by generating far more boxes than the actual number of objects, and then filtering them to retain the appropriate ones. In this way, the number of predicted bounding boxes can be flexibly adjusted, allowing the model to handle both no-object and multi-object scenarios.

Reference: R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.

J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

Problem 2: Optimizers from Scratch.

2.1 Optimizer Implementaiton.

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim  
import torchvision  
import torchvision.transforms as transforms  
import torch.optim as optim  
import matplotlib.pyplot as plt  
import numpy as np  
import math  
import contextlib  
import io
```

Firstly, Define the Optimizers.

1.Momentum.

```
In [2]:  
class MomentumOptimizer:  
    def __init__(self, parameters, lr=0.01, beta=0.9, weight_decay=0.001):  
        self.parameters = list(parameters)  
        self.lr = lr  
        self.beta = beta  
        self.weight_decay = weight_decay  
        self.m = [torch.zeros_like(p) for p in self.parameters]  
  
    def step(self):  
        for i, parameter in enumerate(self.parameters):  
            if parameter.grad is not None:  
                self.m[i] = self.beta * self.m[i] + (parameter.grad + self.weight_decay * torch.sign(parameter.data)) # This is the momentum update.  
                parameter.data -= self.lr * self.m[i] # update the parameters  
  
    def zero_grad(self):  
        for parameter in self.parameters:  
            if parameter.grad is not None:  
                parameter.grad.zero_()
```

2.Nesterov's Accelerated Gradient.

```
In [3]:  
class NAGOptimizer:  
    def __init__(self, parameters, lr=0.001, beta=0.95, weight_decay=0.001):  
        self.parameters = list(parameters)  
        self.lr = lr  
        self.beta = beta  
        self.weight_decay = weight_decay  
        self.t1 = [torch.clone(p).detach() for p in self.parameters] # The Lookahead point_t  
        self.t2 = [torch.zeros_like(p) for p in self.parameters] # The Lookahead point_t+1  
        self.lambda1 = 0 # The Lambda_t  
        self.lambda2 = 0 # The Lambda_t+1  
        self.gamma = 0  
  
    def step(self):  
        for i, parameter in enumerate(self.parameters):  
            if parameter.grad is not None:  
                self.lambda2 = (1 + math.sqrt(1 + 4 * (self.lambda1**2))) / 2 # Update the Lambda and gamma.  
                self.gamma = (1 - self.lambda1) / self.lambda2  
                self.lambda1 = self.lambda2 # Update the Lambda_t, preparing for the next turn.  
  
                grad_with_l1 = parameter.grad + self.weight_decay * torch.sign(parameter.data) # Compute the gradient of l1  
                self.t2[i] = parameter.data - self.lr * grad_with_l1 / self.beta # Update the Lookahead term.  
                parameter.data = (1 - self.gamma) * self.t2[i] + self.gamma * self.t1[i] # Update the parameters.  
                self.t1[i] = self.t2[i].clone().detach() # Update the Lookahead term_t, preparing for the next turn.  
  
    def zero_grad(self):  
        for parameter in self.parameters:  
            if parameter.grad is not None:  
                parameter.grad.zero_()
```

3.RMSprop

```
In [4]: class RMSpropOptimizer:
    def __init__(self, parameters, lr=0.001, beta=0.95, epsilon=1e-8, gamma=1, weight_decay=0.001):
        self.parameters = list(parameters)
        self.lr = lr
        self.beta = beta
        self.weight_decay=weight_decay
        self.epsilon = epsilon
        self.gamma = gamma
        self.v = [torch.zeros_like(p) for p in self.parameters] # The Accumulated Squared Gradient.

    def step(self):
        for i, parameter in enumerate(self.parameters):
            if parameter.grad is not None:
                grad_with_l1 = parameter.grad + self.weight_decay * torch.sign(parameter.data) # Compute the gradient of l

                self.v[i] = (1 - self.beta) * self.v[i] + self.beta * (grad_with_l1 * grad_with_l1) # Update the Accumulated Squared Gradient.

                b_k = self.gamma / (torch.sqrt(self.v[i]) + self.epsilon) # Compute the Preconditioning Term.

                parameter.data -= self.lr * b_k * grad_with_l1 # Update the parameters.

    def zero_grad(self):
        for parameter in self.parameters:
            if parameter.grad is not None:
                parameter.grad.zero_()


```

4.Adam.

```
In [5]: class AdamOptimizer:
    def __init__(self, parameters, lr=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8, weight_decay=0.001):
        self.parameters = list(parameters)
        self.lr = lr
        self.weight_decay = weight_decay
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m1 = [torch.zeros_like(p) for p in self.parameters] # First Order Momentum.
        self.m2 = [torch.zeros_like(p) for p in self.parameters] # Second Order Momentum.
        self.t = 0 # The Step.

    def step(self):
        self.t += 1

        for i, parameter in enumerate(self.parameters):
            if parameter.grad is not None:
                grad_with_l1 = parameter.grad + self.weight_decay * torch.sign(parameter.data) # Compute the gradient of l

                self.m1[i] = self.beta1 * self.m1[i] + (1 - self.beta1) * grad_with_l1 # Update the First Order Momentum.

                self.m2[i] = self.beta2 * self.m2[i] + (1 - self.beta2) * (grad_with_l1 * grad_with_l1) # Update the Second Order Momentum.

                m1_hat = self.m1[i] / (1 - self.beta1 ** self.t) # Debias the First Order Momentum.

                m2_hat = self.m2[i] / (1 - self.beta2 ** self.t) # Debias the Second Order Momentum.

                parameter.data -= self.lr * m1_hat / (torch.sqrt(m2_hat) + self.epsilon) # Update the parameters.

    def zero_grad(self):
        for parameter in self.parameters:
            if parameter.grad is not None:
                parameter.grad.zero_()


```

Secondly, Define the Data, Model and Trainer.

Prepare the datas.

```
In [6]: train_set,test_set,train_loader,test_loader = {},{},{},{}

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.1307,), (0.3081,))])

train_set['mnist'] = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_set['mnist'] = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)

# datas of different batch size.
train_loader0 = torch.utils.data.DataLoader(train_set['mnist'], batch_size=100, shuffle=True, num_workers=0)
test_loader0 = torch.utils.data.DataLoader(test_set['mnist'], batch_size=100, shuffle=False, num_workers=0)
train_loader1 = torch.utils.data.DataLoader(train_set['mnist'], batch_size=4, shuffle=True, num_workers=0)
test_loader1 = torch.utils.data.DataLoader(test_set['mnist'], batch_size=4, shuffle=False, num_workers=0)
train_loader2 = torch.utils.data.DataLoader(train_set['mnist'], batch_size=8, shuffle=True, num_workers=0)
test_loader2 = torch.utils.data.DataLoader(test_set['mnist'], batch_size=8, shuffle=False, num_workers=0)
train_loader3 = torch.utils.data.DataLoader(train_set['mnist'], batch_size=16, shuffle=True, num_workers=0)
test_loader3 = torch.utils.data.DataLoader(test_set['mnist'], batch_size=16, shuffle=False, num_workers=0)
```

```
train_loader4 = torch.utils.data.DataLoader(train_set['mnist'], batch_size=32, shuffle=True, num_workers=0)
test_loader4 = torch.utils.data.DataLoader(test_set['mnist'], batch_size=32, shuffle=False, num_workers=0)
```

Define the model. (Use the example model in discussion section)

```
In [7]: class ExampleNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, 8, 2, 1)
        self.fc1 = nn.Linear(1352, out_channels)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = torch.flatten(x, 1)
        x = self.fc1(x)

    return x
```

Define the trainer. (Use the code present in discussion section)

```
In [8]: def train(model, device, train_loader, criterion, optimizer, epoch, train_losses):
    model.train()
    train_loss = 0
    batch_index = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        batch_index = batch_idx
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        if batch_idx % (len(train_loader)//2) == 0:
            print('Train({})[{:0f}]: Loss: {:.4f}'.format(
                epoch, 100. * batch_idx / len(train_loader), train_loss/(batch_idx+1)))

    train_losses.append(train_loss/(batch_index+1))

def test(model, device, test_loader, criterion, epoch, batch_size, test_losses):
    model.eval()
    test_loss = 0
    correct = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss = (test_loss*batch_size)/len(test_loader.dataset)
    print('Test({}): Loss: {:.4f}, Accuracy: {:.4f}%'.format(
        epoch, test_loss, 100. * correct / len(test_loader.dataset)))
    test_losses.append(test_loss)
```

Some other preparations.

```
In [9]: device = 'cuda'
device = torch.device(device)
num_epochs = 5
in_channels = 1
out_channels = 10
```

Thirdly, Test Whether the Optimizer is works well.

```
In [10]: lr = 0.001
train_losses = []
test_losses = []
batch_size = 100
model = ExampleNet(in_channels, out_channels).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = MomentumOptimizer(model.parameters(), lr)

for epoch in range(1, num_epochs + 1):
    train(model, device, train_loader0, criterion, optimizer, epoch, train_losses)
    test(model, device, test_loader0, criterion, epoch, batch_size, test_losses)
```

```
Train(1)[0%]: Loss: 2.3308
Train(1)[50%]: Loss: 0.7269
Test(1): Loss: 0.3142, Accuracy: 91.0700%
Train(2)[0%]: Loss: 0.3048
Train(2)[50%]: Loss: 0.3233
Test(2): Loss: 0.2825, Accuracy: 92.1400%
Train(3)[0%]: Loss: 0.4576
Train(3)[50%]: Loss: 0.3028
Test(3): Loss: 0.2726, Accuracy: 92.3700%
Train(4)[0%]: Loss: 0.2846
Train(4)[50%]: Loss: 0.2844
Test(4): Loss: 0.2640, Accuracy: 92.6200%
Train(5)[0%]: Loss: 0.3421
Train(5)[50%]: Loss: 0.2754
Test(5): Loss: 0.2512, Accuracy: 92.9600%
```

```
In [11]: lr = 0.001
batch_size = 100
train_losses = []
test_losses = []
model = ExampleNet(in_channels,out_channels).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = NAGOptimizer(model.parameters(), lr)

for epoch in range(1, num_epochs + 1):
    train(model, device, train_loader0, criterion, optimizer, epoch, train_losses)
    test(model, device, test_loader0, criterion, epoch, batch_size, test_losses)
```

```
Train(1)[0%]: Loss: 2.2498
Train(1)[50%]: Loss: 0.5746
Test(1): Loss: 0.1765, Accuracy: 94.8000%
Train(2)[0%]: Loss: 0.1712
Train(2)[50%]: Loss: 0.1895
Test(2): Loss: 0.1783, Accuracy: 94.6600%
Train(3)[0%]: Loss: 0.1918
Train(3)[50%]: Loss: 0.1807
Test(3): Loss: 0.1825, Accuracy: 94.3500%
Train(4)[0%]: Loss: 0.2481
Train(4)[50%]: Loss: 0.1926
Test(4): Loss: 0.1868, Accuracy: 94.2000%
Train(5)[0%]: Loss: 0.2292
Train(5)[50%]: Loss: 0.1925
Test(5): Loss: 0.1836, Accuracy: 94.0200%
```

```
In [12]: lr = 0.001
batch_size = 100
train_losses = []
test_losses = []
model = ExampleNet(in_channels,out_channels).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = RMSpropOptimizer(model.parameters(), lr)

for epoch in range(1, num_epochs + 1):
    train(model, device, train_loader0, criterion, optimizer, epoch, train_losses)
    test(model, device, test_loader0, criterion, epoch, batch_size, test_losses)
```

```
Train(1)[0%]: Loss: 2.2991
Train(1)[50%]: Loss: 0.5209
Test(1): Loss: 0.2739, Accuracy: 92.3800%
Train(2)[0%]: Loss: 0.4226
Train(2)[50%]: Loss: 0.2753
Test(2): Loss: 0.2338, Accuracy: 93.5200%
Train(3)[0%]: Loss: 0.2224
Train(3)[50%]: Loss: 0.2403
Test(3): Loss: 0.2040, Accuracy: 94.3100%
Train(4)[0%]: Loss: 0.2301
Train(4)[50%]: Loss: 0.2147
Test(4): Loss: 0.1927, Accuracy: 94.6300%
Train(5)[0%]: Loss: 0.1626
Train(5)[50%]: Loss: 0.2111
Test(5): Loss: 0.1820, Accuracy: 94.9100%
```

```
In [13]: lr = 0.001
batch_size = 100
train_losses = []
test_losses = []
model = ExampleNet(in_channels,out_channels).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = AdamOptimizer(model.parameters(), lr)

for epoch in range(1, num_epochs + 1):
    train(model, device, train_loader0, criterion, optimizer, epoch, train_losses)
    test(model, device, test_loader0, criterion, epoch, batch_size, test_losses)
```

```

Train(1)[0%]: Loss: 2.3880
Train(1)[50%]: Loss: 0.5182
Test(1): Loss: 0.2197, Accuracy: 94.0200%
Train(2)[0%]: Loss: 0.2132
Train(2)[50%]: Loss: 0.2309
Test(2): Loss: 0.1838, Accuracy: 95.0700%
Train(3)[0%]: Loss: 0.2369
Train(3)[50%]: Loss: 0.1907
Test(3): Loss: 0.1802, Accuracy: 95.0100%
Train(4)[0%]: Loss: 0.2736
Train(4)[50%]: Loss: 0.1723
Test(4): Loss: 0.1509, Accuracy: 95.7900%
Train(5)[0%]: Loss: 0.1244
Train(5)[50%]: Loss: 0.1609
Test(5): Loss: 0.1470, Accuracy: 95.7000%

```

From the results above, we can see that all the classification accuracies under different optimizers are good, which are above 90% after 4 epochs.

(92.9600% for Momentum; 94.0200% for NAG; 94.9100% for RMSprop; 95.7000% for Adam). So the implementation is correct.

2.2 Comparing your optimizer efficiency

Define the super parameters.

```

In [14]: device = 'cuda'
device = torch.device(device)
num_epochs = 5
in_channels = 1
out_channels = 10
criterion = nn.CrossEntropyLoss()

batch_sizes = [4, 8, 16, 32]
learning_rates = [0.001, 0.0005, 0.0001]
train_loaders = [train_loader1, train_loader2, train_loader3, train_loader4]
test_loaders = [test_loader1, test_loader2, test_loader3, test_loader4]

```

Define the train process.

```

In [ ]: train_losses_all = []
test_losses_all = []

for i, batch_size in enumerate(batch_sizes):

    batch_size = batch_size

    for j, lr in enumerate(learning_rates):
        model = ExampleNet(in_channels,out_channels).to(device)
        optimizer = MomentumOptimizer(model.parameters(), lr)
        train_losses = []
        test_losses = []

        with contextlib.redirect_stdout(io.StringIO()):
            for epoch in range(1, num_epochs + 1):
                train(model, device, train_loaders[i], criterion, optimizer, epoch, train_losses)
                test(model, device, test_loaders[i], criterion, epoch, batch_size, test_losses)

        train_losses_all.append(train_losses)
        test_losses_all.append(test_losses)

    for j, lr in enumerate(learning_rates):
        model = ExampleNet(in_channels,out_channels).to(device)
        optimizer = NAGOptimizer(model.parameters(), lr)
        train_losses = []
        test_losses = []

        with contextlib.redirect_stdout(io.StringIO()):
            for epoch in range(1, num_epochs + 1):
                train(model, device, train_loaders[i], criterion, optimizer, epoch, train_losses)
                test(model, device, test_loaders[i], criterion, epoch, batch_size, test_losses)

        train_losses_all.append(train_losses)
        test_losses_all.append(test_losses)

    for j, lr in enumerate(learning_rates):
        model = ExampleNet(in_channels,out_channels).to(device)
        optimizer = RMSpropOptimizer(model.parameters(), lr)
        train_losses = []
        test_losses = []

        with contextlib.redirect_stdout(io.StringIO()):
            for epoch in range(1, num_epochs + 1):
                train(model, device, train_loaders[i], criterion, optimizer, epoch, train_losses)
                test(model, device, test_loaders[i], criterion, epoch, batch_size, test_losses)

        train_losses_all.append(train_losses)
        test_losses_all.append(test_losses)

```

```

with contextlib.redirect_stdout(io.StringIO()):
    for epoch in range(1, num_epochs + 1):
        train(model, device, train_loaders[i], criterion, optimizer, epoch, train_losses)
        test(model, device, test_loaders[i], criterion, epoch, batch_size, test_losses)

    train_losses_all.append(train_losses)
    test_losses_all.append(test_losses)

for j, lr in enumerate(learning_rates):
    model = ExampleNet(in_channels,out_channels).to(device)
    optimizer = AdamOptimizer(model.parameters(), lr)
    train_losses = []
    test_losses = []

    with contextlib.redirect_stdout(io.StringIO()):
        for epoch in range(1, num_epochs + 1):
            train(model, device, train_loaders[i], criterion, optimizer, epoch, train_losses)
            test(model, device, test_loaders[i], criterion, epoch, batch_size, test_losses)

    train_losses_all.append(train_losses)
    test_losses_all.append(test_losses)

print(i)

```

Compare the Train loss under different batch size, optimizers and learning rates.

```

In [17]: batch_sizes = [4, 8, 16, 32]
optimizers = ['Momentum', 'NAG', 'RMSprop', 'Adam']
learning_rates = [0.001, 0.0005, 0.0001]

fig, axs = plt.subplots(4, 4, figsize=(20, 16))

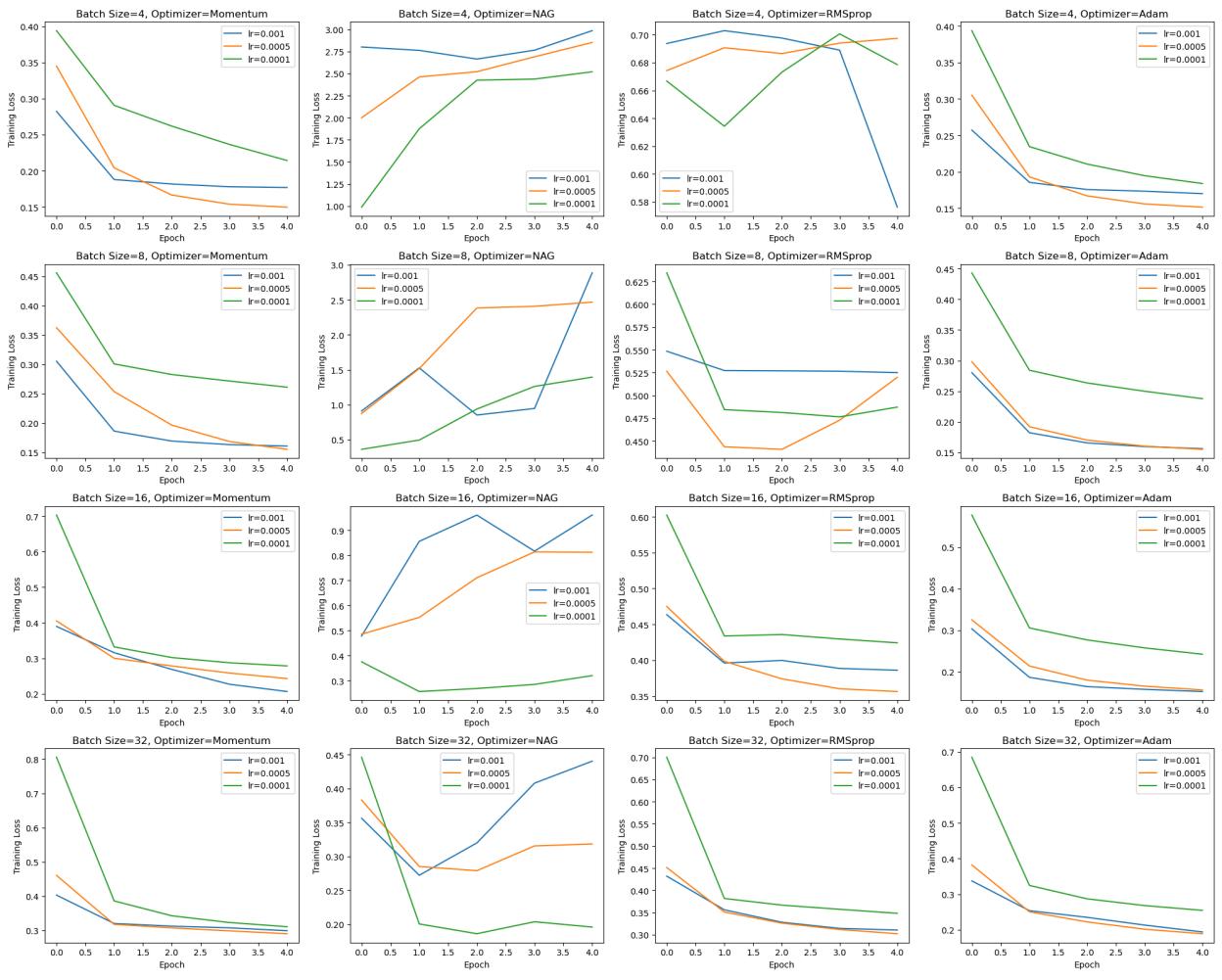
for i, batch_size in enumerate(batch_sizes):
    for j, optimizer in enumerate(optimizers):
        ax = axs[i, j]

        for k, lr in enumerate(learning_rates):
            idx = i * 12 + j * 3 + k
            ax.plot(train_losses_all[idx], label=f"lr={lr}")

        ax.set_title(f"Batch Size={batch_size}, Optimizer={optimizer}")
        ax.set_xlabel("Epoch")
        ax.set_ylabel("Training Loss")
        ax.legend()

plt.tight_layout()
plt.show()

```



Compare the Test loss under different batch size, optimizers and learning rates.

```
In [20]: batch_sizes = [4, 8, 16, 32]
optimizers = ['Momentum', 'NAG', 'RMSprop', 'Adam']
learning_rates = [0.001, 0.0005, 0.0001]

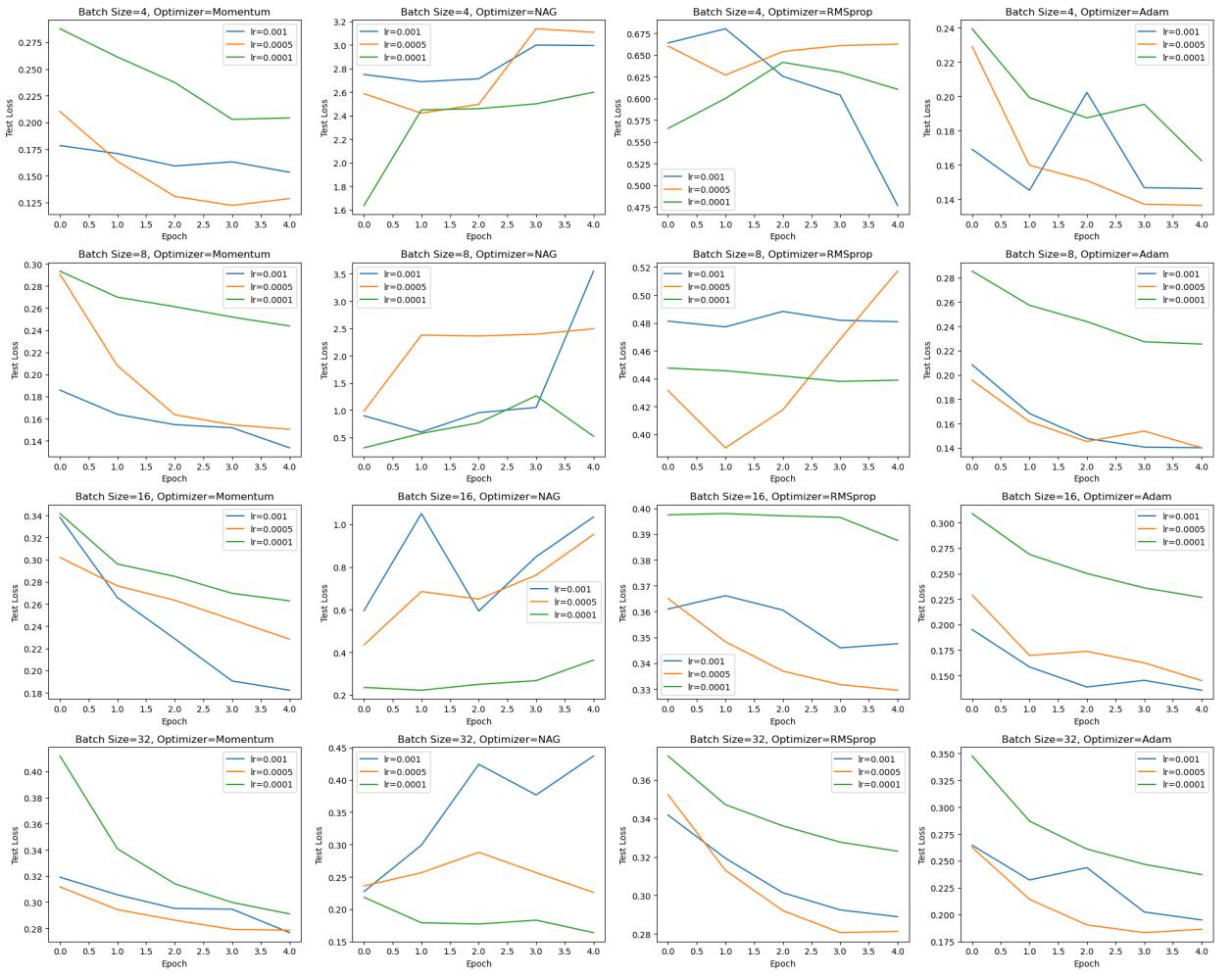
fig, axs = plt.subplots(4, 4, figsize=(20, 16))

for i, batch_size in enumerate(batch_sizes):
    for j, optimizer in enumerate(optimizers):
        ax = axs[i, j]

        for k, lr in enumerate(learning_rates):
            idx = i * 12 + j * 3 + k
            ax.plot(test_losses_all[idx], label=f"lr={lr}")

        ax.set_title(f"Batch Size={batch_size}, Optimizer={optimizer}")
        ax.set_xlabel("Epoch")
        ax.set_ylabel("Test Loss")
        ax.legend()

plt.tight_layout()
plt.show()
```



From the above figures, it can be seen that the NAG algorithm has the worse stability when the batch size is small and the learning rate is high. But it is worth noting that NAG have very good performance under relatively lower learning rate and larger batch size, as shown in figures ($lr=0.0001$; batch size ≥ 16) and part1.

RMSprop shows a medium performance.

In terms of overall performance, the Adam and Momentum algorithms are undoubtedly the best performers, exhibiting high stability across different batch sizes and learning rates, and always achieving the most optimization in the final loss function.

Problem 1

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.optim as optim  
from torch.optim import lr_scheduler  
from torchvision import datasets, transforms  
import matplotlib.pyplot as plt  
import numpy as np  
  
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Define the Sparse Autoencoder, I use MLP as encoder and Dh=x_hat as decoder.

```
In [2]:  
class SparseAutoencoder(nn.Module):  
    def __init__(self, input_size, hidden_size, sparsity_ratio):  
        super(SparseAutoencoder, self).__init__()  
        self.encoder = nn.Sequential(  
            nn.Linear(input_size, 1024),  
            nn.ReLU(),  
            nn.Dropout(0.5),  
            nn.Linear(1024, 1024),  
            nn.ReLU(),  
            nn.Dropout(0.5),  
            nn.Linear(1024, hidden_size)  
        )  
        self.decoder = nn.Linear(hidden_size, input_size)  
        self.sparsity_ratio = sparsity_ratio  
  
    def forward(self, x):  
        encoded = self.encoder(x)  
        decoded = self.decoder(encoded)  
        return decoded, encoded  
  
    def sparse_loss(self, encoded):  
        l1 = torch.mean(torch.abs(encoded))  
        return self.sparsity_ratio * l1
```

Prepare the data.

```
In [3]:  
transform = transforms.Compose([  
    transforms.ToTensor(),  
    transforms.Normalize((0.1307,), (0.3081,)),  
    transforms.Lambda(lambda x: x.view(-1))  
])  
  
batch_size = 10000  
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)  
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)  
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)  
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

Train the model using the MSE as loss and add the L1 of hiddle code as penalty to sparse the h.

```
In [ ]:  
learning_rate = 0.0005  
training_epochs = 200  
input_size = 784  
hidden_size = 1174  
sparsity_ratio = 1.75  
  
model = SparseAutoencoder(input_size, hidden_size, sparsity_ratio)  
model.to(device)  
criterion = nn.MSELoss()  
optimizer = optim.Adam(model.parameters(), lr=learning_rate)  
  
train_loss_list = []  
  
model.train()  
for epoch in range(training_epochs):  
    running_loss = 0  
    for i, (inputs, _) in enumerate(train_loader):  
        inputs = inputs.view(inputs.size(0), -1).to(device)  
        inputs = inputs + torch.randn_like(inputs) * 0.5  
  
        optimizer.zero_grad()  
        outputs, encoded = model(inputs)  
        loss = criterion(outputs, inputs) + model.sparse_loss(encoded)  
        loss.backward()  
        optimizer.step()  
  
        running_loss += loss.item()  
  
    if (epoch + 1) % (training_epochs // 10) == 0:  
        avg_loss = running_loss / len(train_loader)
```

```

        train_loss_list.append(avg_loss)
        print(f'Epoch [{epoch + 1}/{training_epochs}], Loss: {avg_loss:.4f}')

print('Finished Training')

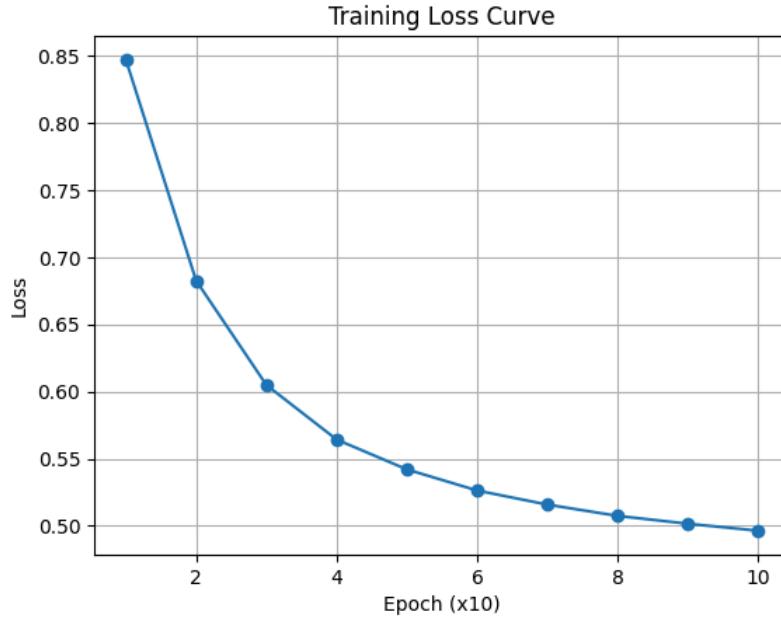
plt.plot(range(1, len(train_loss_list) + 1), train_loss_list, marker='o')
plt.xlabel('Epoch (x10)')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.grid(True)
plt.show()

```

```

Epoch [20/200], Loss: 0.8470
Epoch [40/200], Loss: 0.6825
Epoch [60/200], Loss: 0.6049
Epoch [80/200], Loss: 0.5642
Epoch [100/200], Loss: 0.5420
Epoch [120/200], Loss: 0.5263
Epoch [140/200], Loss: 0.5158
Epoch [160/200], Loss: 0.5075
Epoch [180/200], Loss: 0.5016
Epoch [200/200], Loss: 0.4964
Finished Training

```



Evaluate the performance and plot the comparation of pictures.

```

In [ ]: model.eval()
with torch.no_grad():
    data_iter = iter(test_loader)
    inputs, _ = next(data_iter)
    inputs = inputs.view(inputs.size(0), -1).to(device)

    noise = torch.randn_like(inputs) * 0.4
    noisy_inputs = inputs + noise

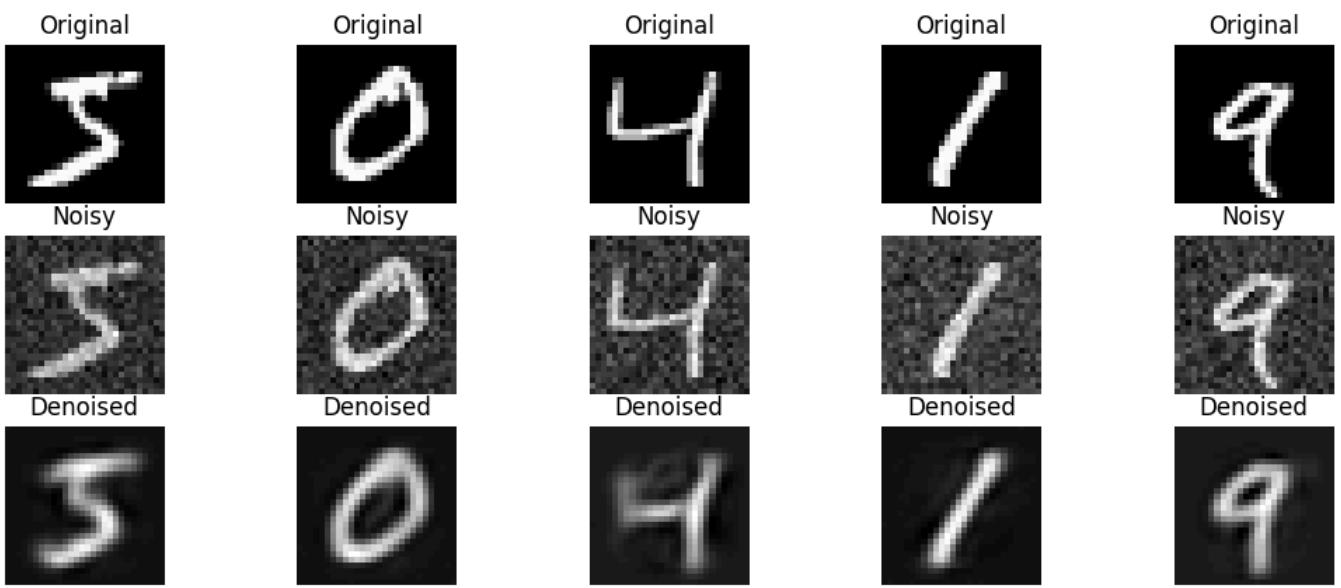
    outputs, encoded = model(noisy_inputs)

f, a = plt.subplots(3, 5, figsize=(13, 5))
for i in range(5):
    a[0][i].imshow(inputs[i].cpu().numpy().reshape(28, 28), cmap='gray')
    a[0][i].axis('off')
    a[0][i].set_title("Original")

    a[1][i].imshow(noisy_inputs[i].cpu().numpy().reshape(28, 28), cmap='gray')
    a[1][i].axis('off')
    a[1][i].set_title("Noisy")

    a[2][i].imshow(outputs[i].cpu().numpy().reshape(28, 28), cmap='gray')
    a[2][i].axis('off')
    a[2][i].set_title("Denoised")
plt.show()

```



Draw the top 5 dictionary vectors for some examples.

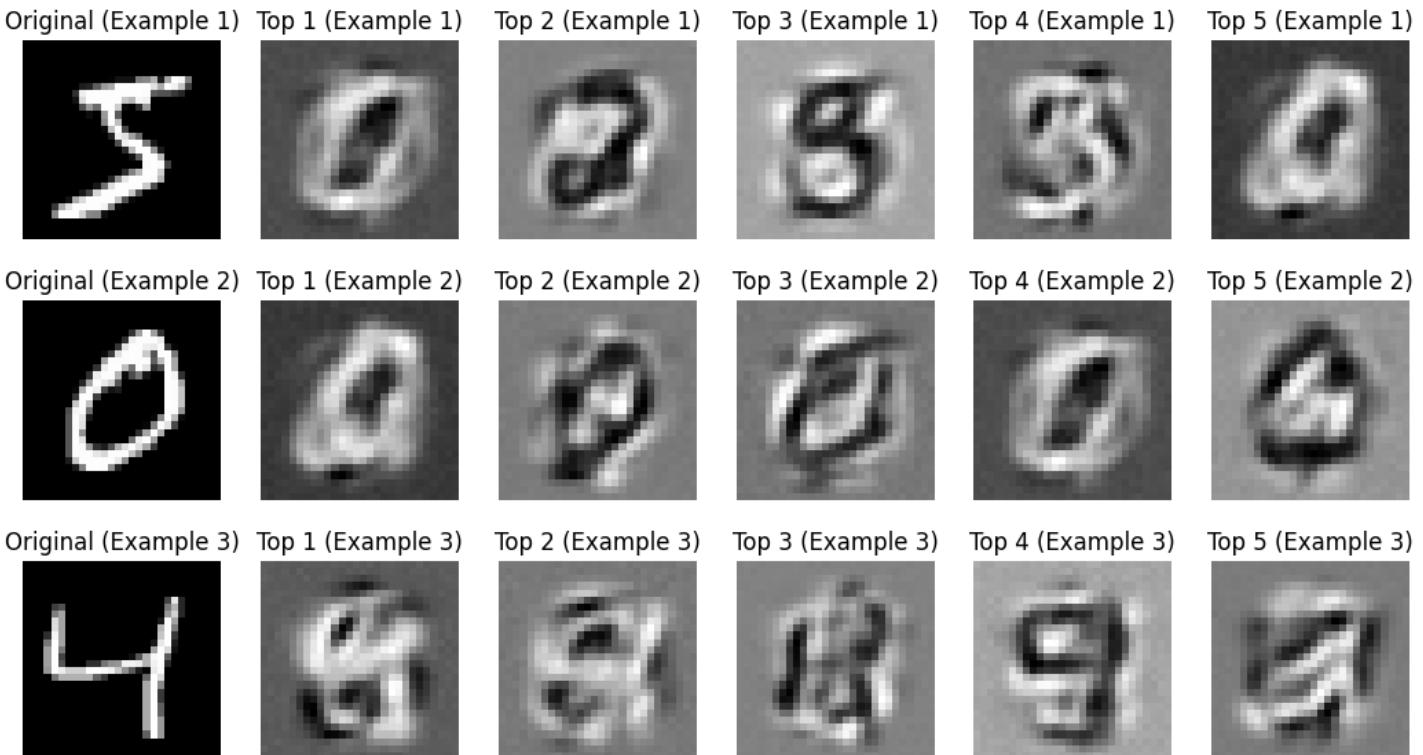
```
In [ ]: D = model.decoder.weight.data.cpu().numpy()

for example_index in range(3):

    h = encoded[example_index].cpu().numpy()
    original_img = inputs[example_index].cpu().numpy().reshape(28, 28) * 0.3081 + 0.1307
    intensities = np.abs(h)

    top_5_indices = np.argsort(intensities)[-5:][::-1]

    f, a = plt.subplots(1, 6, figsize=(13, 5))
    a[0].imshow(original_img, cmap='gray')
    a[0].axis('off')
    a[0].set_title(f'Original (Example {example_index+1})')
    for i, idx in enumerate(top_5_indices):
        dictionary_vector = D[:, idx].reshape(28, 28) * 0.3081 + 0.1307
        a[i+1].imshow(dictionary_vector, cmap='gray')
        a[i+1].axis('off')
        a[i+1].set_title(f'Top {i+1} (Example {example_index+1})')
    plt.show()
```



Problem 2

```
In [1]: import torch
import torch.nn as nn
import torch.optim as optim
from torch.distributions import Normal
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from IPython.display import clear_output
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Prepare the data.

```
In [ ]: batch_size = 1024

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda x: x.view(-1))
])

train_loader = torch.utils.data.DataLoader(datasets.MNIST('./data', train=True, download=True, transform=transform), batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(datasets.MNIST('./data', train=False, transform=transform), batch_size=batch_size, shuffle=True)
```

Define the Model as well as the loss function.

PPCA Setup

The generative model of PPCA is given by:

$$x = Wz + \mu + \epsilon$$

where $W \in \mathbb{R}^{D \times K}$, $z \sim N(0, I)$, and $\epsilon \sim N(0, \Psi)$, $\Psi = \text{diag}(\sigma_1^2, \dots, \sigma_D^2)$.

Loss Function

The negative log-likelihood loss function is given by:

$$\mathcal{L}_{\text{loss}}(W, \Psi) = -\text{likelihood}(W, \Psi) = \frac{N}{2} \log |\Sigma| + \frac{1}{2} \sum_{i=1}^N (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) + \text{Constant}$$

where $\Sigma = WW^T + \Psi$.

Posterior of Latent Variables

The posterior mean of the latent variable z_i given an observation x_i is:

$$z_i^{\text{mean}} = \mathbb{E}[z_i | x_i] = M^{-1}W^T\Psi^{-1}(x_i - \mu)$$

where:

$$M = I + W^T\Psi^{-1}W$$

Reconstruction of x

The reconstruction of x_i from the posterior mean of z_i is:

$$\hat{x} = Wz_i^{\text{mean}} + \mu = WM^{-1}W^T\Psi^{-1}(x_i - \mu) + \mu$$

```
In [ ]: class PPCA(nn.Module):
    def __init__(self, data_dim, latent_dim):
        super(PPCA, self).__init__()
        self.data_dim = data_dim
        self.latent_dim = latent_dim

        self.W = nn.Parameter(torch.randn(data_dim, latent_dim) * 0.01)
        self.log_sigma_sq = nn.Parameter(torch.zeros(data_dim))

    def forward(self, x):
        Psi_inv = torch.diag(torch.exp(-self.log_sigma_sq)).to(device)
        Wt_Psi_inv = self.W.t() @ Psi_inv
        M_inv = torch.inverse(torch.eye(self.latent_dim, device=device) + Wt_Psi_inv @ self.W)
        z_mean = M_inv @ Wt_Psi_inv @ (x - x.mean(dim=0)).t()
        z_mean = z_mean.t()
        x_reconstructed = z_mean @ self.W.t() + x.mean(dim=0)

    return x_reconstructed

    def loss_function(self, x):
        Wt_W = self.W @ self.W.t()
        Psi = torch.diag(torch.exp(self.log_sigma_sq))
```

```

Sigma = Wt + Psi
log_det_Sigma = torch.logdet(Sigma)

Psi_inv = torch.diag(torch.exp(-self.log_sigma_sq))
Sigma_inv = Psi_inv - Psi_inv @ self.W @ torch.inverse(torch.eye(self.latent_dim, device=device) + self.W.t() @ Psi_inv @ self.W) @ self.W.t()

second_term = torch.sum((x - x.mean(dim=0)) @ Sigma_inv * (x - x.mean(dim=0)), dim=1)
loss = 0.5 * (log_det_Sigma + second_term.mean())

return loss

```

Train the Models.

```

In [ ]: latent_dims = [2, 4, 8, 32]
all_train_losses = []
trained_models = []
learning_rate = 0.1
num_epochs = 500

def train_ppca(model, train_loader, train_losses, epochs=20, lr=0.01):
    optimizer = optim.Adam(model.parameters(), lr=lr)
    scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[2, 10, 20, 40, 50, 70, 90, 110, 130, 150, 180, 200, 300, 400], gamma=0.1)

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for batch_idx, (data, _) in enumerate(train_loader):
            data = data.to(device)
            optimizer.zero_grad()
            loss = model.loss_function(data)
            loss.backward()
            optimizer.step()
            scheduler.step()
        running_loss += loss.item()
        train_losses.append(running_loss/len(train_loader))
    print("Training Complete!")

    return model

for latent_dim in latent_dims:
    model = PPCA(data_dim=28*28, latent_dim=latent_dim).to(device)
    save_path = f"ppca_model_latent_dim_{latent_dim}.pth"
    train_losses = []

    trained_model = train_ppca(model, train_loader, train_losses, epochs=num_epochs, lr=learning_rate)
    all_train_losses.append(train_losses)
    trained_models.append(trained_model)
    torch.save(trained_model.state_dict(), save_path)

```

Training Complete!
Training Complete!
Training Complete!
Training Complete!

Plot the loss curve.

```

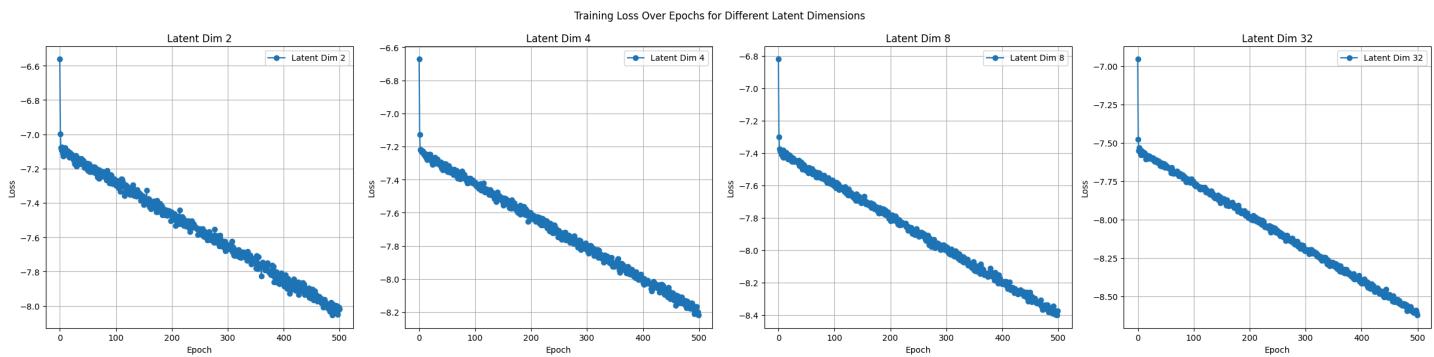
In [6]: def plot_all_train_losses_in_row(all_train_losses, latent_dims):
    fig, axes = plt.subplots(1, 4, figsize=(24, 6))
    fig.suptitle('Training Loss Over Epochs for Different Latent Dimensions')

    for idx, (train_losses, latent_dim) in enumerate(zip(all_train_losses, latent_dims)):
        axes[idx].plot(train_losses, label=f'Latent Dim {latent_dim}', marker='o')
        axes[idx].set_xlabel('Epoch')
        axes[idx].set_ylabel('Loss')
        axes[idx].set_title(f'Latent Dim {latent_dim}')
        axes[idx].legend()
        axes[idx].grid(True)

    plt.tight_layout()
    plt.subplots_adjust(top=0.88)
    plt.show()

plot_all_train_losses_in_row(all_train_losses, latent_dims)

```



Generate new random samples with PPCA with the model weights obtained.

```
In [ ]: def generate_samples_with_noise(models, train_loader, num_samples=5):
    num_models = len(models)
    fig, axes = plt.subplots(num_models, num_samples, figsize=(num_samples * 2, num_models * 2))

    for model in models:
        model.eval()

    with torch.no_grad():
        data_means = []
        for data, _ in train_loader:
            data = data.view(data.size(0), -1).to(device)
            data_means.append(data.mean(dim=0))
        mu = torch.stack(data_means).mean(dim=0).to(device)

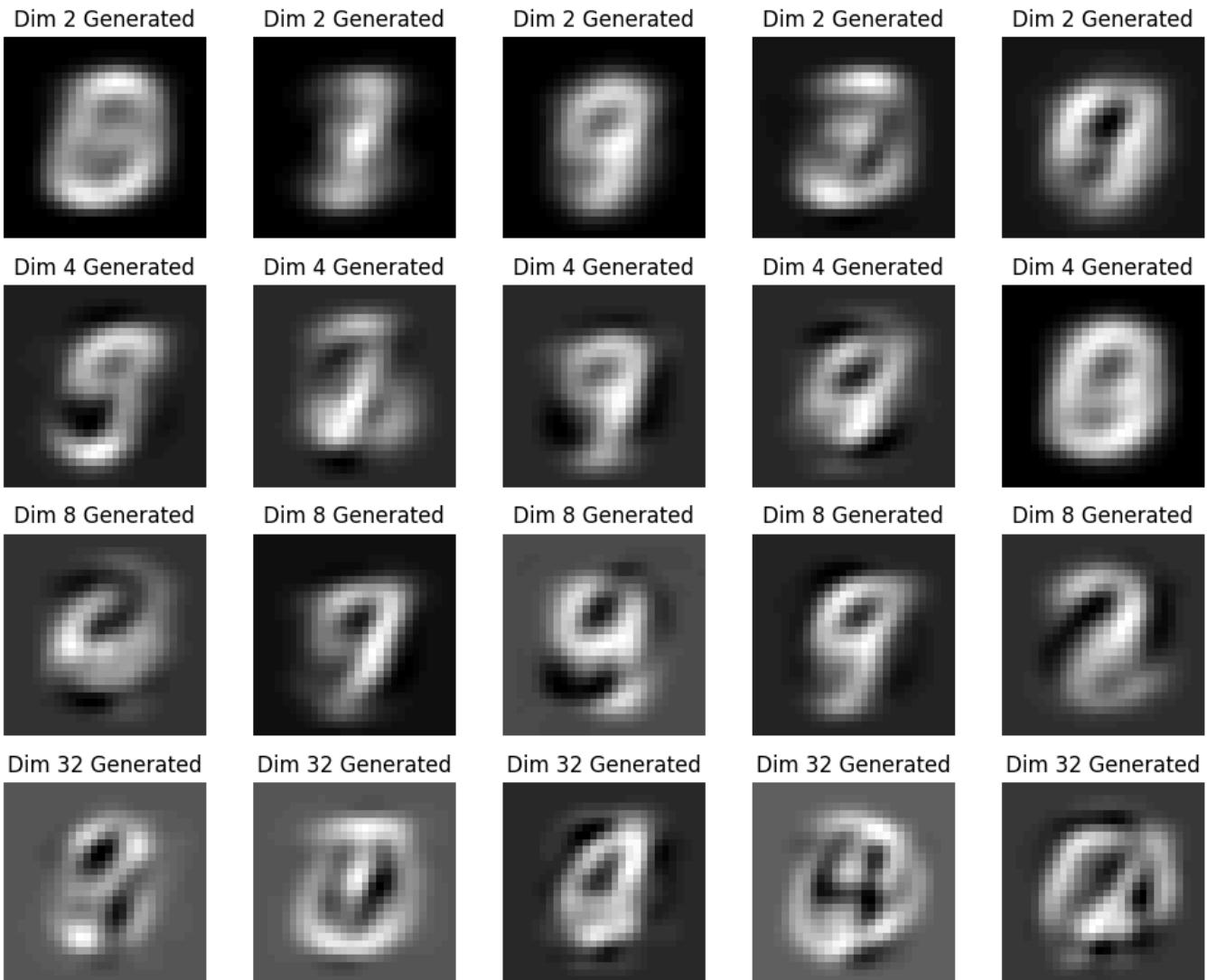
        for model_idx, model in enumerate(models):
            std_devs = torch.sqrt(torch.exp(model.log_sigma_sq)).to(device)
            sampled_z = torch.randn(num_samples, model.latent_dim).to(device)
            sigma_noise = torch.randn(num_samples, model.data_dim).to(device) * std_devs

            generated_samples = torch.mm(sampled_z, model.W.t()) + mu
            generated_samples_with_noise = generated_samples + sigma_noise

            for i in range(num_samples):
                axes[model_idx, i].imshow(generated_samples[i].view(28, 28).cpu().numpy(), cmap='gray')
                axes[model_idx, i].set_title(f'Dim {latent_dims[model_idx]} Generated')
                axes[model_idx, i].axis('off')

    plt.tight_layout()
    plt.show()

generate_samples_with_noise(trained_models, train_loader, num_samples=5)
```



Show the reconstruction effect using the Posterior of Latent Variables.

For showing the correctness of implementation.

```
In [12]: def visualize_reconstruction(models, test_loader, num_images=5):
    num_models = len(models)
    fig, axes = plt.subplots(num_models + 1, num_images, figsize=(num_images * 2, (num_models + 1) * 2))

    for model in models:
        model.eval()
```

```

with torch.no_grad():
    data, _ = next(iter(test_loader))
    data = data.view(data.size(0), -1).to(device)

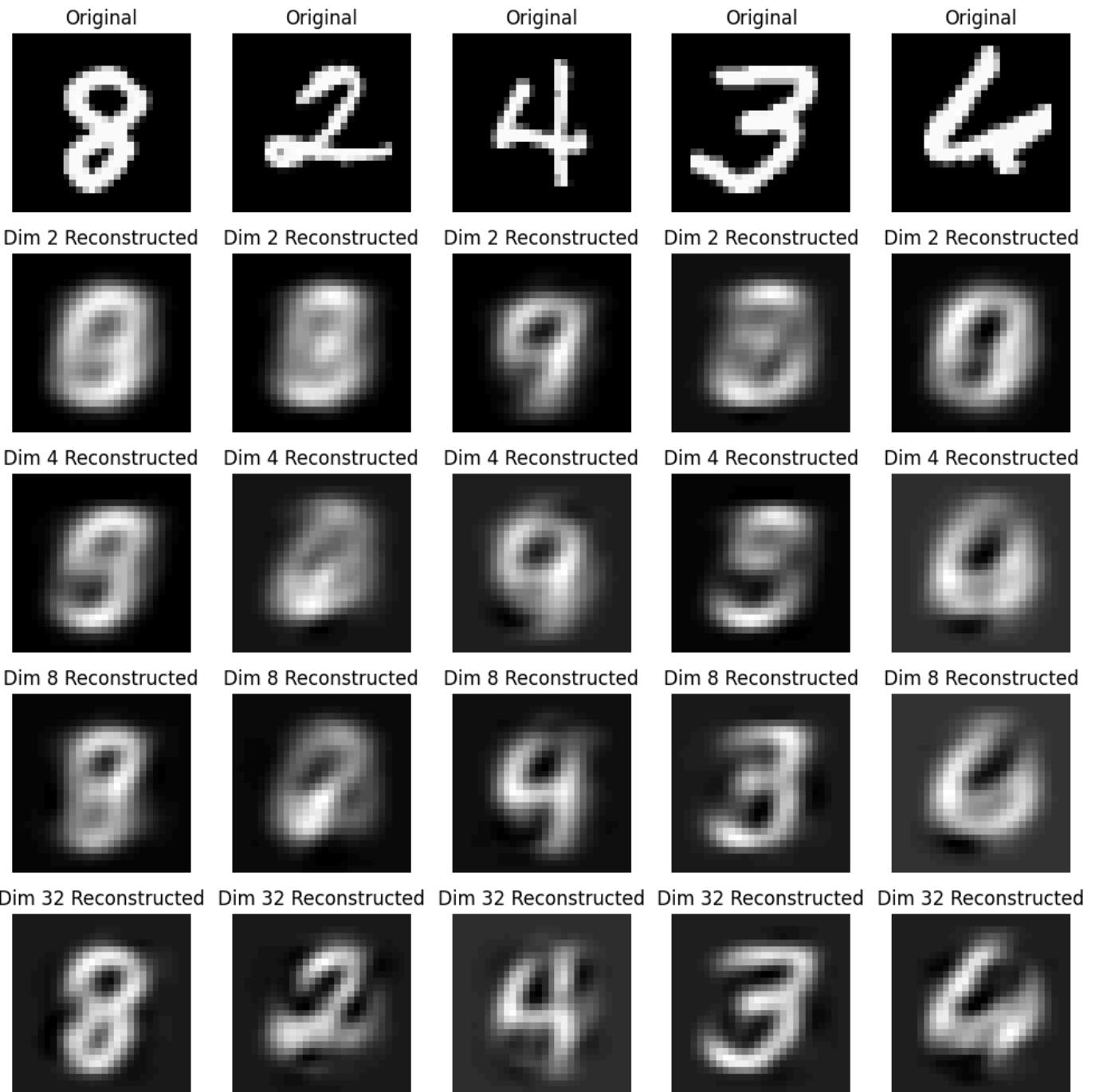
    for i in range(num_images):
        axes[0, i].imshow(data[i].view(28, 28).to('cpu').numpy(), cmap='gray')
        axes[0, i].set_title('Original')
        axes[0, i].axis('off')

    for model_idx, model in enumerate(models):
        x_hat = model(data)
        for i in range(num_images):
            axes[model_idx + 1, i].imshow(x_hat[i].view(28, 28).to('cpu').numpy(), cmap='gray')
            axes[model_idx + 1, i].set_title(f'Dim {latent_dims[model_idx]} Reconstructed')
            axes[model_idx + 1, i].axis('off')

plt.tight_layout()
plt.show()

visualize_reconstruction(trained_models, test_loader, num_images=5)

```



Problem 3

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
from torch.optim import lr_scheduler  
from torch.utils.data import DataLoader  
from torchvision import datasets, transforms, models  
import matplotlib.pyplot as plt  
import numpy as np  
import os  
  
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Define the AutoEncoder.

```
In [ ]:  
class ConvAutoEncoder(nn.Module):  
    def __init__(self):  
        super(ConvAutoEncoder, self).__init__()  
  
        self.Encoder = nn.Sequential(  
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),  
            nn.ReLU(True),  
            nn.Conv2d(32, 48, kernel_size=3, stride=2, padding=1),  
            nn.ReLU(True),  
            nn.Conv2d(48, 96, kernel_size=3, stride=2, padding=1),  
            nn.ReLU(True),  
            nn.Conv2d(96, 126, kernel_size=3, stride=2, padding=1), # 8x8x96 -> 4x4x164  
            nn.ReLU(True)  
        )  
        # After encoder, X encoded from 3x32x32(3072) to 4x4x164(2624).  
        self.Decoder = nn.Sequential(  
            nn.ConvTranspose2d(126, 96, kernel_size=3, stride=2, padding=1, output_padding=1),  
            nn.ReLU(True),  
            nn.ConvTranspose2d(96, 48, kernel_size=3, stride=2, padding=1, output_padding=1),  
            nn.ReLU(True),  
            nn.ConvTranspose2d(48, 32, kernel_size=3, stride=2, padding=1, output_padding=1),  
            nn.ReLU(True),  
            nn.ConvTranspose2d(32, 3, kernel_size=3, stride=1, padding=1),  
            nn.Sigmoid()  
        )  
  
    def forward(self, x):  
        x = self.Encoder(x)  
        feature = x  
  
        x = self.Decoder(x)  
  
        return x, feature
```

Prepare the Data.

```
In [4]:  
batch_size = 128  
num_epoch = 100  
  
transform = transforms.Compose([  
    transforms.ToTensor(),  
])  
  
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)  
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)  
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)  
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Files already downloaded and verified
Files already downloaded and verified

Train the Model.

```
In [5]:  
trained_model = ConvAutoEncoder().to(device)  
criterion = nn.MSELoss()  
min_loss = 10e+10  
optimizer = optim.Adam(trained_model.parameters(), lr= 1e-3, weight_decay= 1e-4)  
scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)  
  
train_loss_list = []  
  
for epoch in range(num_epoch):  
    train_loss = 0  
  
    for batch_ind, (image, _) in enumerate(train_loader):  
        image = image.to(device)  
        optimizer.zero_grad()  
        image_out, _ = trained_model(image)  
        loss = criterion(image_out, image)
```

```

loss.backward()
optimizer.step()
train_loss += loss.item()

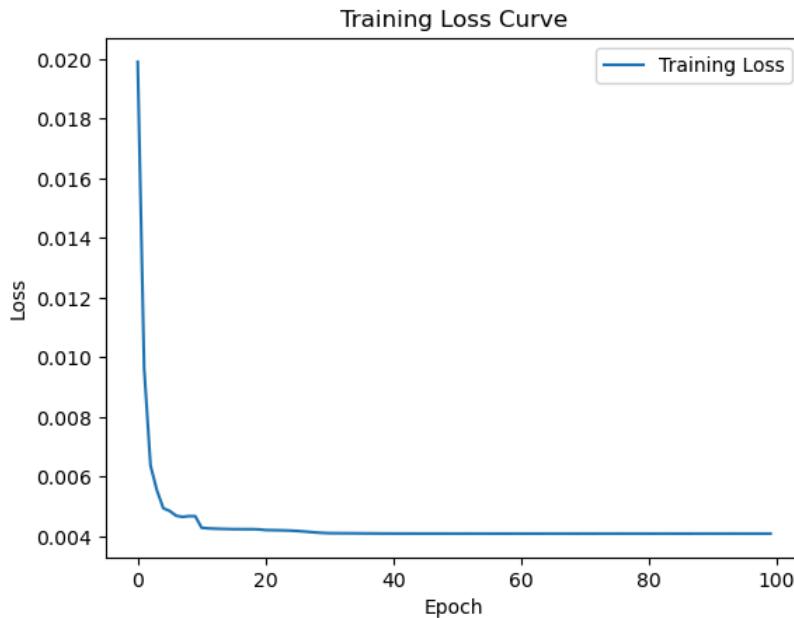
scheduler.step()

avg_train_loss = train_loss / (batch_ind + 1)
train_loss_list.append(avg_train_loss)

if avg_train_loss < min_loss:
    torch.save(trained_model.state_dict(), 'ConvAE_cifar10.pth')
    min_loss = avg_train_loss

plt.plot(range(num_epoch), train_loss_list, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()

```



ConvAE_model_random is the model with initial weights and ConvAE_model_trained is the model loaded the trained weights.

```
In [6]: ConvAE_model_random = ConvAutoEncoder().to(device)
ConvAE_model_trained = ConvAutoEncoder().to(device)
ConvAE_model_trained.load_state_dict(torch.load('ConvAE_cifar10.pth'))
```

C:\Users\14857\AppData\Local\Temp\ipykernel_5520\838969938.py:3: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

```
ConvAE_model_trained.load_state_dict(torch.load('ConvAE_cifar10.pth'))
```

```
Out[6]: <All keys matched successfully>
```

Show the reconstruction effect.

```
In [7]: ConvAE_model_trained.eval()
images, _ = next(iter(train_loader))
images = images.to(device)

with torch.no_grad():
    reconstructed1, _ = ConvAE_model_random(images)
    reconstructed2, _ = ConvAE_model_trained(images)

images = images.cpu()
reconstructed1 = reconstructed1.cpu()
reconstructed2 = reconstructed2.cpu()

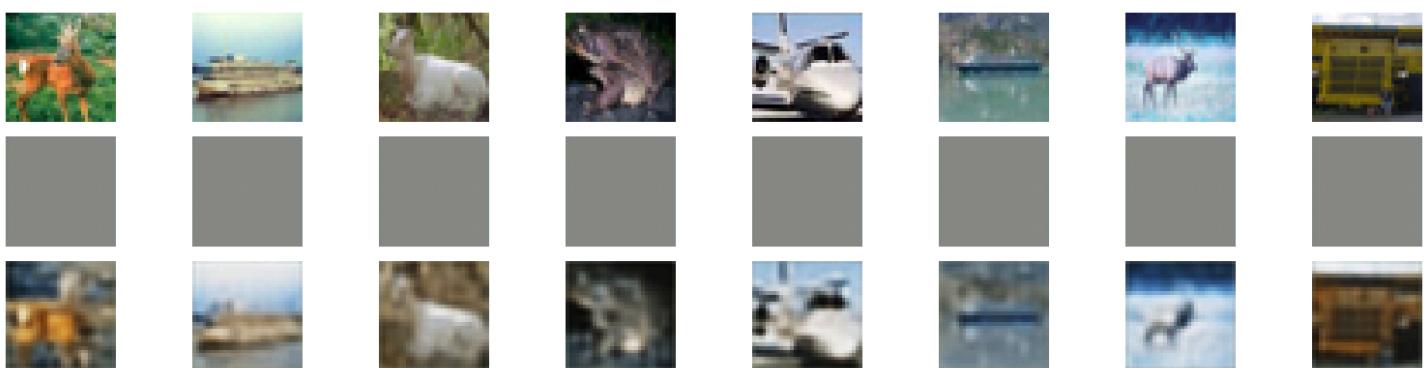
fig, axes = plt.subplots(3, 8, figsize=(16, 4))

for i in range(8):
    axes[0, i].imshow(images[i].permute(1, 2, 0).numpy())
    axes[0, i].axis('off')

    axes[1, i].imshow(reconstructed1[i].permute(1, 2, 0).numpy())
    axes[1, i].axis('off')

    axes[2, i].imshow(reconstructed2[i].permute(1, 2, 0).numpy())
    axes[2, i].axis('off')
```

```
plt.tight_layout()  
plt.show()
```



Instead of Save all the H0 and H1 features, I choose to define the classifier and CombinedModel which combine the autoencoder and classifier. Since I freeze the parameters of the autoencoder, so it is equivalent to save H0/H1 first and then train the classifier.

So the X will firstly be encoded to H0/H1 and then H0/H1 be directly transported to classifier.

```
In [ ]: class Classifier(nn.Module):  
    def __init__(self):  
        super(Classifier, self).__init__()  
  
        self.features = nn.Sequential(  
            nn.Conv2d(126, 256, kernel_size=3, stride=1, padding=1),  
            nn.BatchNorm2d(256),  
            nn.LeakyReLU(0.01, inplace=True),  
  
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),  
            nn.BatchNorm2d(512),  
            nn.LeakyReLU(0.01, inplace=True),  
  
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),  
            nn.BatchNorm2d(512),  
            nn.LeakyReLU(0.01, inplace=True),  
  
            nn.MaxPool2d(2, stride=2)  
        )  
  
        self.classifier = nn.Sequential(  
            nn.AdaptiveAvgPool2d((1, 1)),  
            nn.Flatten(),  
            nn.Linear(512, 512),  
            nn.LeakyReLU(0.01, inplace=True),  
            nn.Dropout(0.5),  
            nn.Linear(512, 256),  
            nn.LeakyReLU(0.01, inplace=True),  
            nn.Dropout(0.5),  
            nn.Linear(256, 10)  
        )  
  
    def forward(self, x):  
        x = self.features(x)  
        x = self.classifier(x)  
        return x
```

```
In [ ]: class CombinedModel(nn.Module):  
    def __init__(self, autoencoder):  
        super(CombinedModel, self).__init__()  
        self.autoencoder = autoencoder  
        for param in self.autoencoder.parameters():  
            param.requires_grad = False  
        self.classifier = Classifier()  
  
    def forward(self, x):  
        _, feature = self.autoencoder(x)  
        output = self.classifier(feature)  
        return output
```

Train the model with H0 features.

```
In [14]: model_H0 = CombinedModel(ConvAE_model_random).to(device)  
criterion = nn.CrossEntropyLoss()  
min_loss = float('inf')  
optimizer = optim.Adam(model_H0.classifier.parameters(), lr=1e-3, weight_decay=1e-4)  
scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)  
  
train_loss_list = []  
  
for epoch in range(50):  
    model_H0.train()
```

```

train_loss = 0

for batch_ind, (image, label) in enumerate(train_loader):
    image = image.to(device)
    label = label.long().to(device)

    optimizer.zero_grad()
    y_hat = model_H0(image)
    loss = criterion(y_hat, label)

    loss.backward()
    optimizer.step()

    train_loss += loss.item()

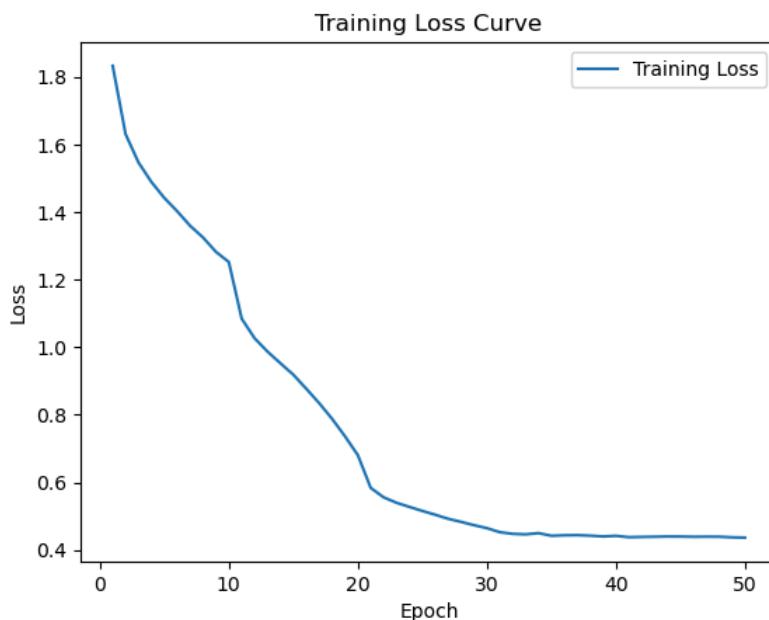
scheduler.step()

avg_train_loss = train_loss / (batch_ind + 1)
train_loss_list.append(avg_train_loss)

if avg_train_loss < min_loss:
    torch.save(model_H0.state_dict(), 'random_model.pth')
    min_loss = avg_train_loss

plt.plot(range(1, 51), train_loss_list, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()

```



Train the model with H1 features.

```

In [15]: model_H1 = CombinedModel(ConvAE_model_trained).to(device)
criterion = nn.CrossEntropyLoss()
min_loss = float('inf')
optimizer = optim.Adam(model_H1.classifier.parameters(), lr=1e-3, weight_decay=1e-4)
scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

train_loss_list = []

for epoch in range(50):
    model_H1.train()
    train_loss = 0

    for batch_ind, (image, label) in enumerate(train_loader):
        image = image.to(device)
        label = label.long().to(device)

        optimizer.zero_grad()
        y_hat = model_H1(image)
        loss = criterion(y_hat, label)

        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    scheduler.step()

    avg_train_loss = train_loss / (batch_ind + 1)
    train_loss_list.append(avg_train_loss)

    if avg_train_loss < min_loss:
        torch.save(model_H1.state_dict(), 'trained_model.pth')

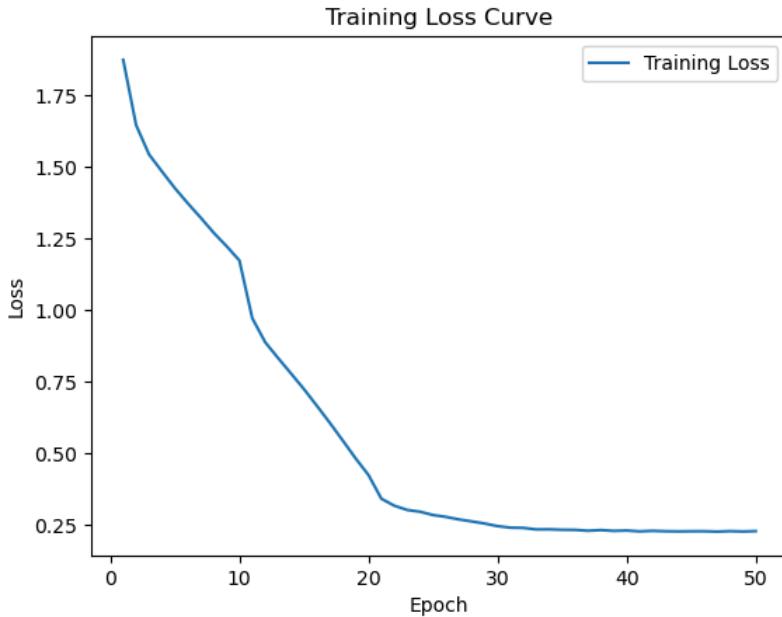
```

```

min_loss = avg_train_loss

plt.plot(range(1, 51), train_loss_list, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()

```



```

In [16]: clf_0 = CombinedModel(ConvAE_model_random)
clf_0.load_state_dict(torch.load('random_model.pth'))
clf_1 = CombinedModel(ConvAE_model_trained)
clf_1.load_state_dict(torch.load('trained_model.pth'))
clf_0.eval()
clf_0.to(device)
clf_1.eval()
clf_1.to(device)

```

C:\Users\14857\AppData\Local\Temp\ipykernel_5520\3682011334.py:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

clf_0.load_state_dict(torch.load('random_model.pth'))
C:\Users\14857\AppData\Local\Temp\ipykernel_5520\3682011334.py:4: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
clf_1.load_state_dict(torch.load('trained_model.pth'))

```

Out[16]: CombinedModel(
    (autoencoder): ConvAutoEncoder(
        (Encoder): Sequential(
            (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU(inplace=True)
            (2): Conv2d(32, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (3): ReLU(inplace=True)
            (4): Conv2d(48, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (5): ReLU(inplace=True)
            (6): Conv2d(96, 126, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
            (7): ReLU(inplace=True)
        )
        (Decoder): Sequential(
            (0): ConvTranspose2d(126, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
            (1): ReLU(inplace=True)
            (2): ConvTranspose2d(96, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
            (3): ReLU(inplace=True)
            (4): ConvTranspose2d(48, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
            (5): ReLU(inplace=True)
            (6): ConvTranspose2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (7): Sigmoid()
        )
    )
    (classifier): ClassifierFromFeature(
        (features): Sequential(
            (0): Conv2d(126, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): LeakyReLU(negative_slope=0.01, inplace=True)
            (3): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (5): LeakyReLU(negative_slope=0.01, inplace=True)
            (6): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (7): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (8): LeakyReLU(negative_slope=0.01, inplace=True)
            (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        )
        (classifier): Sequential(
            (0): AdaptiveAvgPool2d(output_size=(1, 1))
            (1): Flatten(start_dim=1, end_dim=-1)
            (2): Linear(in_features=512, out_features=512, bias=True)
            (3): LeakyReLU(negative_slope=0.01, inplace=True)
            (4): Dropout(p=0.5, inplace=False)
            (5): Linear(in_features=512, out_features=256, bias=True)
            (6): LeakyReLU(negative_slope=0.01, inplace=True)
            (7): Dropout(p=0.5, inplace=False)
            (8): Linear(in_features=256, out_features=10, bias=True)
        )
    )
)
)

```

Test the performance of two models.

```

In [17]: def test_clf(model, device, data, target):
    model.eval()
    correct = 0
    with torch.no_grad():
        data, target = data.to(device), target.to(device)
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
    return 100. * correct / len(data)

```

```

In [18]: iter_test = iter(test_loader)
image_test, label_test = next(iter_test)
image_test = image_test.to(device)
acc_random = test_clf(clf_0, device, image_test, label_test)
acc_trained = test_clf(clf_1, device, image_test, label_test)
print('Classification accuracy for Model with H0 as input: {}'.format(acc_random))
print('Classification accuracy for Model with H1 as input: {}'.format(acc_trained))

```

```

Classification accuracy for Model with H0 as input: 87.5
Classification accuracy for Model with H1 as input: 96.875

```

The results show that the model trained with the extracted features H1 achieves a classification accuracy of 96.875%, which is higher than the training accuracy of 87.5% obtained with the randomly mapped features H0. The key difference here is that H0 is a random mapping of the image; thus, while it retains some image information, it may capture more non-essential information and lose important details. In contrast, H1, although used for image reconstruction and subject to some information loss, is more likely to retain the more critical aspects of the image. Therefore, in terms of the importance of extracted information, H1 is superior to H0, leading to better performance when using H1 for classification.

Problem 1: GAN

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.optim as optim  
from torch.optim import lr_scheduler  
from torchvision import datasets, transforms  
import matplotlib.pyplot as plt  
import numpy as np  
from torch.utils.data import DataLoader, SubsetRandomSampler  
  
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

1.1 Implement a Deep Convolutional GAN (DCGAN)

Prepare the data.

```
In [2]:  
transform = transforms.Compose([  
    transforms.ToTensor(),  
    transforms.Normalize([0.5], [0.5]),  
])  
  
batch_size = 1000  
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)  
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)  
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)  
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

```
In [3]:  
latent_dim = 100  
img_shape = (1, 28, 28)  
lr = 0.0002  
n_epochs = 400
```

Define the generator.

```
In [4]:  
class Generator(nn.Module):  
    def __init__(self):  
        super(Generator, self).__init__()  
        self.model = nn.Sequential(  
            nn.ConvTranspose2d(latent_dim, 256, kernel_size=7, stride=1, padding=0),  
            nn.BatchNorm2d(256),  
            nn.ReLU(inplace=True),  
            nn.Dropout(0.3),  
  
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),  
            nn.BatchNorm2d(128),  
            nn.ReLU(inplace=True),  
            nn.Dropout(0.3),  
  
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),  
            nn.BatchNorm2d(64),  
            nn.ReLU(inplace=True),  
            nn.Dropout(0.3),  
  
            nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),  
            nn.BatchNorm2d(32),  
            nn.ReLU(inplace=True),  
            nn.Dropout(0.3),  
  
            nn.Conv2d(32, 1, kernel_size=3, stride=1, padding=1),  
            nn.Tanh()  
        )  
  
    def forward(self, z):  
        z = z.view(z.size(0), latent_dim, 1, 1)  
        img = self.model(z)  
        return img
```

Define the discriminator.

```
In [5]:  
class Discriminator(nn.Module):  
    def __init__(self):  
        super(Discriminator, self).__init__()  
        self.model = nn.Sequential(  
            nn.Conv2d(1, 64, kernel_size=4, stride=2, padding=1),  
            nn.BatchNorm2d(64),  
            nn.LeakyReLU(0.2, inplace=True),  
            nn.Dropout(0.3),  
  
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),  
            nn.BatchNorm2d(128),  
            nn.LeakyReLU(0.2, inplace=True),  
            nn.Dropout(0.3),
```

```

        nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
        nn.BatchNorm2d(256),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Dropout(0.3),

        nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
        nn.BatchNorm2d(512),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Dropout(0.3),

        nn.AdaptiveAvgPool2d((4, 4))
    )

    self.fc = nn.Sequential(
        nn.Flatten(),
        nn.Linear(512 * 4 * 4, 1),
        nn.Sigmoid()
)

```

```

def forward(self, img):
    features = self.model(img)
    validity = self.fc(features)
    return validity

```

```

In [6]: generator = Generator().to(device)
discriminator = Discriminator().to(device)

adversarial_loss = nn.BCELoss()
optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))

```

Train the model.

```

In [7]: for epoch in range(n_epochs):
    for i, (imgs, _) in enumerate(train_loader):
        real_imgs = imgs.to(device)
        batch_size = real_imgs.size(0)

        valid = torch.ones(batch_size, 1, device=device)
        fake = torch.zeros(batch_size, 1, device=device)

        optimizer_G.zero_grad()
        z = torch.randn(batch_size, latent_dim, device=device)
        gen_imgs = generator(z)
        g_loss = adversarial_loss(discriminator(gen_imgs), valid)
        g_loss.backward()
        optimizer_G.step()

        optimizer_D.zero_grad()
        real_loss = adversarial_loss(discriminator(real_imgs), valid)
        fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
        d_loss = (real_loss + fake_loss) / 2
        d_loss.backward()
        optimizer_D.step()

    if (epoch + 1) % 40 == 0:
        print(f"[Epoch {epoch + 1}/{n_epochs}] [D loss: {d_loss.item():.4f}] [G loss: {g_loss.item():.4f}]")

torch.save(generator.state_dict(), "generator.pth")
torch.save(discriminator.state_dict(), "discriminator.pth")

```

```

/root/miniconda3/lib/python3.12/site-packages/torch/nn/modules/conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
    return F.conv2d(input, weight, bias, self.stride,
/root/miniconda3/lib/python3.12/site-packages/torch/autograd/graph.py:744: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
    return Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
[Epoch 40/400] [D loss: 0.4796] [G loss: 1.7630]
[Epoch 80/400] [D loss: 0.3972] [G loss: 1.2897]
[Epoch 120/400] [D loss: 0.5719] [G loss: 0.7195]
[Epoch 160/400] [D loss: 0.2084] [G loss: 2.9183]
[Epoch 200/400] [D loss: 0.2013] [G loss: 2.9095]
[Epoch 240/400] [D loss: 0.1873] [G loss: 1.6038]
[Epoch 280/400] [D loss: 0.5124] [G loss: 1.0463]
[Epoch 320/400] [D loss: 2.7404] [G loss: 0.0113]
[Epoch 360/400] [D loss: 0.0579] [G loss: 3.8257]
[Epoch 400/400] [D loss: 0.1779] [G loss: 4.9219]

```

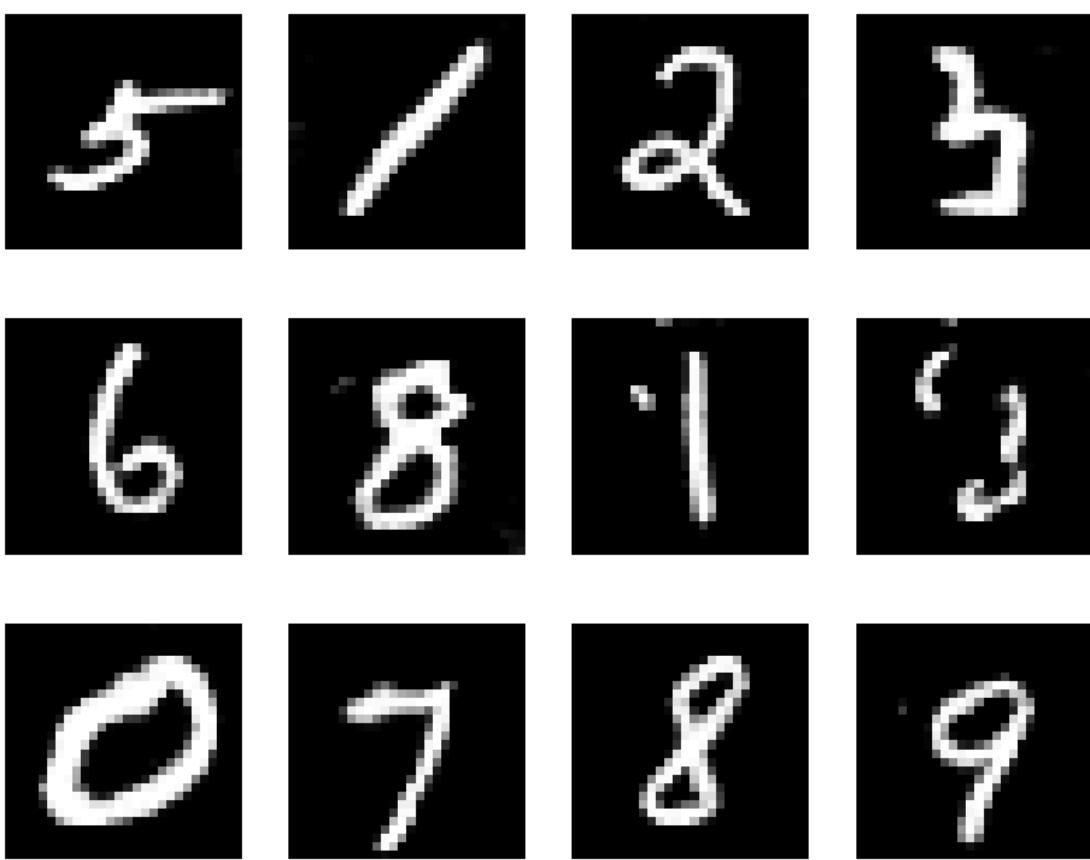
Utilize the generator to produce 12 new samples

```

In [8]: z = torch.randn(12, latent_dim, device=device)
gen_imgs = generator(z).cpu().detach()

fig, axs = plt.subplots(3, 4, figsize=(10, 8))
for i, ax in enumerate(axs.flatten()):
    ax.imshow(gen_imgs[i].squeeze(), cmap='gray')
    ax.axis('off')
plt.show()

```



1.2 GAN as a pre-training framework

Removing the final linear layer of your discriminator to form a feature extractor.

```
In [9]: class FeatureExtractor(nn.Module):
    def __init__(self, discriminator):
        super(FeatureExtractor, self).__init__()
        self.features = nn.Sequential(*list(discriminator.model.children()))

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        return x

discriminator = Discriminator().to(device)
discriminator.load_state_dict(torch.load("discriminator.pth"))
feature_extractor = FeatureExtractor(discriminator).to(device)
feature_extractor.eval()
```

```
Out[9]: FeatureExtractor(
    (features): Sequential(
        (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): LeakyReLU(negative_slope=0.2, inplace=True)
        (3): Dropout(p=0.3, inplace=False)
        (4): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (6): LeakyReLU(negative_slope=0.2, inplace=True)
        (7): Dropout(p=0.3, inplace=False)
        (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (10): LeakyReLU(negative_slope=0.2, inplace=True)
        (11): Dropout(p=0.3, inplace=False)
        (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (14): LeakyReLU(negative_slope=0.2, inplace=True)
        (15): Dropout(p=0.3, inplace=False)
        (16): AdaptiveAvgPool2d(output_size=(4, 4))
    )
)
```

Define single linear layer classifier.

```
In [10]: class LinearClassifier(nn.Module):
    def __init__(self, input_dim, num_classes=10):
        super(LinearClassifier, self).__init__()
        self.classifier = nn.Linear(input_dim, num_classes)

    def forward(self, x):
        return self.classifier(x)
```

10% of the training set images.

```
In [11]: dataset_size = len(train_loader.dataset)
indices = np.random.choice(dataset_size, size=int(0.1 * dataset_size), replace=False)
sampler = SubsetRandomSampler(indices)
subset_loader = DataLoader(train_loader.dataset, batch_size=64, sampler=sampler)
```

```
In [12]: data, labels = [], []
for imgs, lbls in subset_loader:
    features = feature_extractor(imgs.to(device)).detach().cpu()
    data.append(features)
    labels.append(lbls)

train_features = torch.cat(data)
train_labels = torch.cat(labels)
```

Train the simple classifier.

```
In [13]: input_dim = 512 * 4 * 4
classifier = LinearClassifier(input_dim).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(classifier.parameters(), lr=0.001)

n_epochs = 1000
for epoch in range(n_epochs):
    classifier.train()
    optimizer.zero_grad()
    outputs = classifier(train_features.to(device))
    loss = criterion(outputs, train_labels.to(device))
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 100 == 0:
        print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {loss.item():.4f}")
```

```
Epoch [100/1000], Loss: 0.1285
Epoch [200/1000], Loss: 0.0680
Epoch [300/1000], Loss: 0.0433
Epoch [400/1000], Loss: 0.0302
Epoch [500/1000], Loss: 0.0223
Epoch [600/1000], Loss: 0.0171
Epoch [700/1000], Loss: 0.0136
Epoch [800/1000], Loss: 0.0111
Epoch [900/1000], Loss: 0.0093
Epoch [1000/1000], Loss: 0.0079
```

Report both the training and testing performance

```
In [14]: def evaluate(classifier, loader, feature_extractor):
    classifier.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for imgs, labels in loader:
            features = feature_extractor(imgs.to(device)).detach()
            outputs = classifier(features)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted.cpu() == labels).sum().item()
    accuracy = 100 * correct / total
    return accuracy

train_accuracy = evaluate(classifier, subset_loader, feature_extractor)
print(f"Train Set Accuracy: {train_accuracy:.2f}%")
test_accuracy = evaluate(classifier, test_loader, feature_extractor)
print(f"Test Set Accuracy: {test_accuracy:.2f}%")
```

```
Train Set Accuracy: 100.00%
Test Set Accuracy: 97.16%
```

The results demonstrate that the GAN's discriminator effectively captures meaningful features from the images, enabling the linear classifier to achieve excellent performance with the high training accuracy (100%) and strong test accuracy (97.16%).

Training a GAN discriminator to differentiate between real and fake images helps it learn detailed and meaningful features of the data distribution. Even the discriminator's task requires distinguishing real data from the generated data which is not same as classifier, it focuses on high-level representations and subtle patterns in the input images.

Problem 2.1

part 1

Given the probability $P(v_i|h)$ for the Gaussian-Bernoulli Restricted Boltzmann Machine (GBRBM):

$$P(v_i|h) = P(v_i|v_{-i}, h) = \frac{P(v_i, v_{-i}, h)}{P(v_{-i}, h)},$$

we can write it as:

$$P(v_i|h) = \frac{\exp(-E(v, h))/Z}{\int_{v_i} \exp(-E(v|h))/Z dv_i},$$

where $E(v, h)$ is the energy function, and Z is the partition function.

After simplifying the expression, we have:

$$P(v_i|h) = \frac{\exp\left(\sum_k \sum_j \frac{v_k}{\sigma_k} w_{kj} h_j + \sum_j a_j h_j - \sum_k \frac{(v_k - b_k)^2}{2\sigma_k^2}\right)}{\int_{v_i} \exp\left(\sum_k \sum_j \frac{v_k}{\sigma_k} w_{kj} h_j + \sum_j a_j h_j - \sum_k \frac{(v_k - b_k)^2}{2\sigma_k^2}\right) dv_i}.$$

Define:

$$\hat{A} = \sum_{k \neq i} \sum_j \frac{v_k}{\sigma_k} w_{kj} h_j + \sum_j a_j h_j - \sum_{k \neq i} \frac{(v_k - b_k)^2}{2\sigma_k^2}.$$

Thus, we can rewrite the denominator as:

$$\exp(\hat{A}) \int_{v_i} \exp\left(\frac{v_i}{\sigma_i} \sum W_{ij} h_j - \frac{(v_i - b_i)^2}{2\sigma_i^2}\right) dv_i.$$

The numerator becomes:

$$\exp(\hat{A}) \cdot \exp\left(\frac{v_i}{\sigma_i} \sum W_{ij} h_j - \frac{(v_i - b_i)^2}{2\sigma_i^2}\right).$$

Therefore:

$$P(v_i|h) = \frac{\exp\left(\frac{v_i}{\sigma_i} \sum W_{ij} h_j - \frac{(v_i - b_i)^2}{2\sigma_i^2}\right)}{\int_{v_i} \exp\left(\frac{v_i}{\sigma_i} \sum W_{ij} h_j - \frac{(v_i - b_i)^2}{2\sigma_i^2}\right) dv_i}.$$

And we have:

$$\frac{v_i}{\sigma_i} \sum W_{ij} h_j - \frac{(v_i - b_i)^2}{2\sigma_i^2} = \frac{-[v_i - (\sigma_i \sum_j W_{ij} h_j + b_i)]}{2\sigma_i^2} + \left[\frac{(\sigma_i \sum_j W_{ij} h_j + b_i)^2}{2\sigma_i^2} - \frac{b_i^2}{2\sigma_i^2}\right]$$

Define:

$$B = \left[\frac{(\sigma_i \sum_j W_{ij} h_j + b_i)^2}{2\sigma_i^2} - \frac{b_i^2}{2\sigma_i^2}\right]$$

And define:

$$\mu = \sigma_i \sum_j W_{ij} h_j + b_i.$$

So:

$$P(v_i|h) = \frac{\exp(B) \exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right)}{\exp(B) \int_{v_i} \exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right) dv_i} = \frac{\exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right)}{\int_{v_i} \exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right) dv_i}$$

And we have:

$$\int_{v_i} \exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right) dv_i = \sqrt{2\pi} \sigma_i,$$

Further simplifying, we have:

$$P(v_i|h) = \frac{\exp\left(-\frac{(v_i - \mu)^2}{2\sigma_i^2}\right)}{\sqrt{2\pi} \sigma_i},$$

Thus:

$$P(v_i|h) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2\sigma_i^2} \left[v_i - (b_i + \sigma_i \sum_j W_{ij} h_j)\right]^2\right),$$

which is a Gaussian distribution:

$$v_i|h \sim \mathcal{N}(b_i + \sigma_i \sum_j W_{ij} h_j, \sigma_i^2).$$

part 2

The conditional probability $P(h_j|v)$ can be expressed as:

$$P(h_j|v) = P(h_j|h_{-j}, v) = \frac{P(h_j, h_{-j}, v)}{P(h_j = 1, h_{-j}, v) + P(h_j = 0, h_{-j}, v)}$$

The energy function $E(v, h)$ is given by:

$$E(v, h) = - \left(h_j \sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j h_j + \sum_{k \neq j} \sum_i W_{ik} \frac{v_i}{\sigma_i} h_k + \sum_{k \neq j} \alpha_k h_k - \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} \right)$$

Define:

$$A = \sum_{k \neq j} \sum_i W_{ik} \frac{v_i}{\sigma_i} h_k + \sum_{k \neq j} \alpha_k h_k - \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2}$$

Thus, we have:

$$P(h_j|v) = \frac{\exp(A) \cdot \exp\left(h_j \sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j h_j\right) / Z}{\exp(A) \cdot \exp\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right) / Z + \exp(A) \cdot \exp(0) / Z}$$

Simplifying further:

$$P(h_j|v) = \frac{\exp\left(h_j \sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right)}{1 + \exp\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right)}$$

Thus:

$$P(h_j = 1|v) = \frac{\exp\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right)}{1 + \exp\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right)} = \text{sigmoid}\left(\sum_i W_{ij} \frac{v_i}{\sigma_i} + \alpha_j\right)$$

Problem 2: Gaussian-Bernoulli Restricted Boltzmann Machines

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim  
import torch.optim as optim  
  
import torchvision  
import torchvision.transforms as transforms  
from torchvision.utils import make_grid  
  
import matplotlib.pyplot as plt  
import numpy as np
```

2.2 GB-RBM on KMNIST

Prepare the data.

```
In [2]:  
device = 'cuda'  
batch_size = 128  
transform = transforms.Compose(  
    [transforms.ToTensor(),  
     ])  
  
train_set = torchvision.datasets.KMNIST(root='./data', train=True, download=True, transform=transform)  
test_set = torchvision.datasets.KMNIST(root='./data', train=False, download=True, transform=transform)  
  
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=0)  
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)
```

Define the model. Since we will normalize the data, so set $\sigma=1$.

```
In [3]:  
class RBM(nn.Module):  
    """Restricted Boltzmann Machine template."""  
  
    def __init__(self, D: int, F: int, k: int):  
        """Creates an instance RBM module.  
        Args:  
            D: Size of the input data.  
            F: Size of the hidden variable.  
            k: Number of MCMC iterations for negative sampling.  
        The function initializes the weight (W) and biases (c & b).  
        """  
        super().__init__()  
        self.W = nn.Parameter(torch.randn(F, D) * 1e-2)  
        self.c = nn.Parameter(torch.zeros(D))  
        self.b = nn.Parameter(torch.zeros(F))  
        self.k = k  
  
    def sample(self, p):  
        """Sample from a Bernoulli distribution defined by a given parameter."""  
        return torch.bernoulli(p)  
  
    def sample_gaussian(self, p):  
        """Sample from a Gaussian distribution defined by a given parameter."""  
        return p + torch.randn_like(p)  
  
    def P_h_x(self, x):  
        """Returns the conditional P(h|x)."""  
        return torch.sigmoid(F.linear(x, self.W, self.b))  
  
    def P_x_h(self, h):  
        """Returns the conditional P(x|h)."""  
        return F.linear(h, self.W.t(), self.c)  
  
    def free_energy(self, x):  
        """Returns the Average F(x) free energy. (Slide 11, Lecture 14)."""  
        vbias_term = (-(x-self.c)*(x-self.c)/2).sum(dim=1)  
        wv_b = F.linear(x, self.W, self.b)  
        hidden_term = F.softplus(wv_b).sum(dim=1)  
        return (-hidden_term - vbias_term).mean()  
  
    def forward(self, v):  
        """Performs forward propagation through the model."""  
        v_sampled = v  
        for _ in range(self.k):  
            p_h_given_v = self.P_h_x(v_sampled)  
            h_sampled = self.sample(p_h_given_v)  
  
            p_v_given_h = self.P_x_h(h_sampled)  
            v_sampled = self.sample_gaussian(p_v_given_h)  
  
            p_h_given_v = self.P_h_x(v_sampled)  
            h_sampled = self.sample(p_h_given_v)
```

```
    return v_sampled, p_v_given_h
```

Define the train process.

```
In [4]: def train(model, device, train_loader, optimizer, epoch):
    model.train()
    train_loss = 0
    for data, _ in train_loader:
        data = data.view(data.size(0), -1).to(device)

        # Normalizing data
        mean, std = data.mean(), data.std()
        data = (data - mean) / std

        v_sampled, p_v_given_h = model(data)

        optimizer.zero_grad()
        loss = model.free_energy(data) - model.free_energy(v_sampled)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_loss /= len(train_loader.dataset)
    if epoch%5==0:
        print('Train Epoch: {} \tLoss: {:.6f}'.format(epoch, train_loss))
```

Train the 3 models with latent dim = [16, 64, 256].

```
In [5]: seed = 42
num_epochs = 25
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
M = [16, 64, 256]
models = []
optimizers = []

for i in range(3):
    rbm = RBM(D=28 * 28 * 1, F=M[i], k=5).to(device)
    optimizer = optim.Adam(rbm.parameters(), lr=1e-3)
    models.append(rbm)
    optimizers.append(optimizer)

for i, (rbm, optimizer) in enumerate(zip(models, optimizers)):
    print("Training model {} with M = {}".format(i + 1, M[i]))
    for epoch in range(1, num_epochs + 1):
        train(rbm, device, train_loader, optimizer, epoch)
```

```
Training model 1 with M = 16:
Train Epoch: 5 Loss: -1.125684
Train Epoch: 10 Loss: -1.172729
Train Epoch: 15 Loss: -1.188537
Train Epoch: 20 Loss: -1.197791
Train Epoch: 25 Loss: -1.205249
Training model 2 with M = 64:
Train Epoch: 5 Loss: -1.674580
Train Epoch: 10 Loss: -1.767623
Train Epoch: 15 Loss: -1.812097
Train Epoch: 20 Loss: -1.843711
Train Epoch: 25 Loss: -1.866311
Training model 3 with M = 256:
Train Epoch: 5 Loss: -2.171796
Train Epoch: 10 Loss: -2.285825
Train Epoch: 15 Loss: -2.363311
Train Epoch: 20 Loss: -2.421793
Train Epoch: 25 Loss: -2.465338
```

Define the plot function.

```
In [6]: def show(img1, img2, M):
    npimg1 = img1.cpu().numpy()
    npimg2 = img2.cpu().numpy()

    fig, axes = plt.subplots(1, 2, figsize=(20, 10))

    axes[0].imshow(np.transpose(npimg1, (1, 2, 0)), interpolation='nearest')
    axes[0].set_title("Original Image", fontsize=16)
    axes[0].axis('off')

    axes[1].imshow(np.transpose(npimg2, (1, 2, 0)), interpolation='nearest')
    axes[1].set_title(f"Reconstructed Image (M={M})", fontsize=16)
    axes[1].axis('off')

    plt.tight_layout()
    plt.show()
```

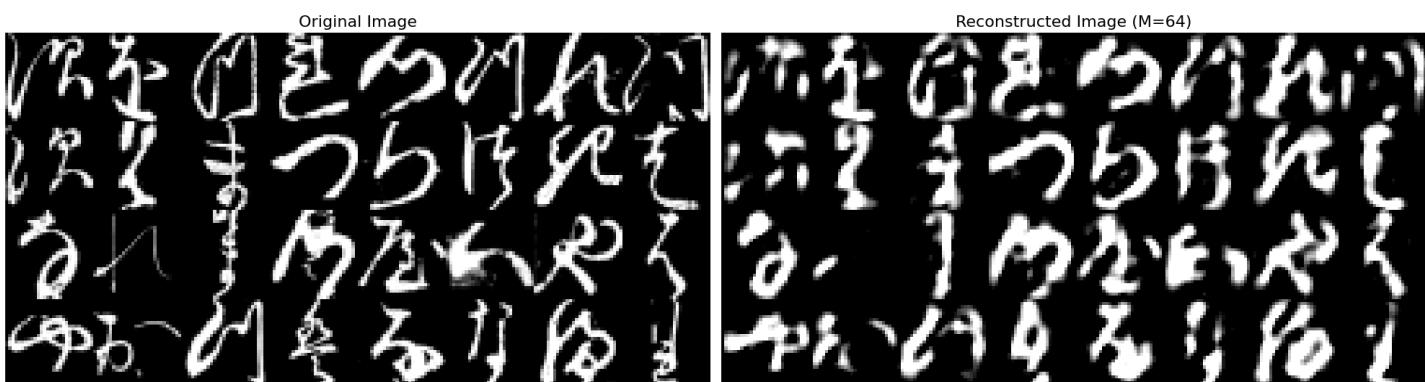
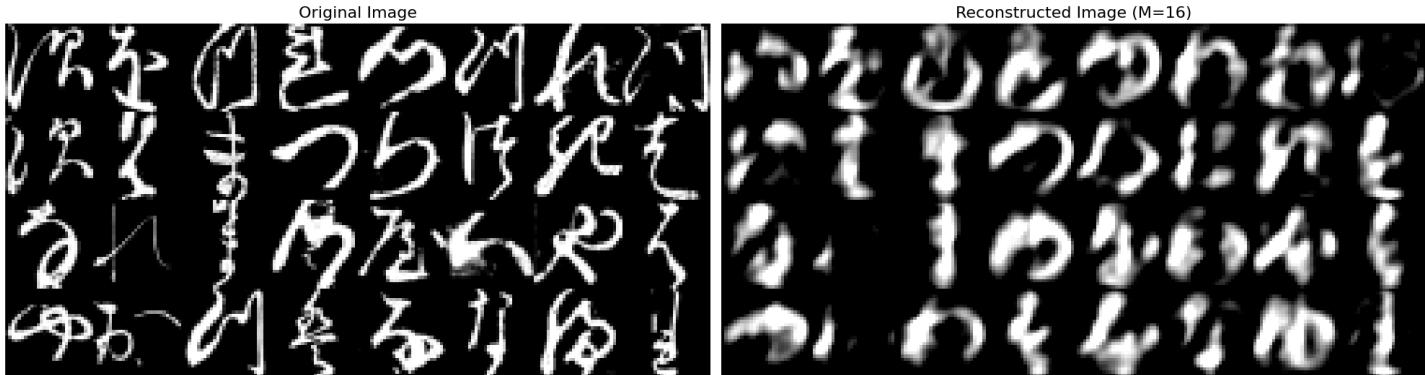
Show the images in the train set.

```
In [7]: train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=False, num_workers=0)
for i in range(3):
    rbm = models[i]
    data, _ = next(iter(train_loader))
    data = data[:32]
    data_size = data.size()
    data = data.view(data.size(0), -1).to(device)
    mean, std = data.mean(), data.std()
    bdata = (data - mean) / std
    vh_x = rbm.P_h_x(bdata)
    vx_h = rbm.P_x_h(vh_x)
    vx_h = vx_h.detach()
    show(make_grid(data.reshape(data_size), padding=0), make_grid(vx_h.reshape(data_size).clip(min=0,max=1), padding=0), M[i])
    plt.show()
```



Show the images in the test set.

```
In [8]: for i in range(3):
    rbm = models[i]
    data, _ = next(iter(test_loader))
    data = data[:32]
    data_size = data.size()
    data = data.view(data.size(0), -1).to(device)
    mean, std = data.mean(), data.std()
    bdata = (data - mean) / std
    vh_x = rbm.P_h_x(bdata)
    vx_h = rbm.P_x_h(vh_x)
    vx_h = vx_h.detach()
    show(make_grid(data.reshape(data_size), padding=0), make_grid(vx_h.reshape(data_size).clip(min=0,max=1), padding=0), M[i])
    plt.show()
```



Compute and report the mean squared reconstruction error (MSE) for inferences.

```
In [13]: def test(model, device, data_loader):
    model.eval()
    total_loss = 0
    total_samples = 0

    mse_loss = nn.MSELoss()

    with torch.no_grad():
        for data, _ in data_loader:
            data = data.view(data.size(0), -1).to(device)

            mean, std = data.mean(), data.std()
            bdata = (data - mean) / std
            vh_x = model.P_h_x(bdata)
            vx_h = model.P_x_h(vh_x)

            loss = mse_loss(vx_h, bdata)
            total_loss += loss.item() * data.size(0)
            total_samples += data.size(0)

    average_loss = total_loss / total_samples
    return average_loss
```

```
In [14]: for i in range(3):
    model = models[i]
    print('model {} with M = {}'.format(i + 1, M[i]))
    average_loss = test(model, device, train_loader)
    print('Average MSE Loss in Train Set: {:.4f}'.format(average_loss))
    average_loss = test(model, device, test_loader)
    print('Average MSE Loss in Test Set: {:.4f}'.format(average_loss))
```

model 1 with M = 16
 Average MSE Loss in Train Set: 0.5964
 Average MSE Loss in Test Set: 0.6304
 model 2 with M = 64
 Average MSE Loss in Train Set: 0.3595
 Average MSE Loss in Test Set: 0.3900
 model 3 with M = 256
 Average MSE Loss in Train Set: 0.1446
 Average MSE Loss in Test Set: 0.1637

Problem 3: Two Variational Autoencoders

```
In [1]:  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim  
import torch.optim as optim  
  
import torchvision  
import torchvision.transforms as transforms  
from torchvision.utils import make_grid  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader, random_split  
  
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.manifold import TSNE  
  
device = 'cuda'
```

3.1 Vanilla VAE and 3.2 C-VAE

Define the VAE, c is the conditional information and in Vanilla VAE, just put c=0.

```
In [ ]:  
class VAE(nn.Module):  
    def __init__(self, n_in, n_hid, z_dim):  
        super().__init__()  
  
        self.fc1 = nn.Linear(n_in + 10, n_hid)  
        self.fc21 = nn.Linear(n_hid, z_dim)  
        self.fc22 = nn.Linear(n_hid, z_dim)  
  
        self.fc3 = nn.Linear(z_dim + 10, n_hid)  
        self.fc4 = nn.Linear(n_hid, n_in)  
  
    def encode(self, x, c):  
        """Encoder forward pass."""  
        x = torch.cat((x, c), dim=1)  
        h1 = F.relu(self.fc1(x))  
        mu = self.fc21(h1)  
        logvar = self.fc22(h1)  
        return mu, logvar  
  
    def reparameterize(self, mu, logvar):  
        """Implements: z = mu + epsilon * stdev."""  
        std = torch.exp(0.5 * logvar)  
        eps = torch.randn_like(std)  
        z = mu + eps * std  
        return z  
  
    def decode(self, z, c):  
        """Decoder forward pass."""  
        z = torch.cat((z, c), dim=1)  
        h3 = F.relu(self.fc3(z))  
        recon_x = torch.sigmoid(self.fc4(h3))  
        return recon_x  
  
    def forward(self, x, c):  
        """Defines the complete forward pass of the VAE."""  
        mu, logvar = self.encode(x, c)  
        z = self.reparameterize(mu, logvar)  
        recon_x = self.decode(z, c)  
        return recon_x, mu, logvar
```

Prepare the data.

```
In [ ]:  
batch_size = 128  
  
transform = transforms.Compose([transforms.ToTensor(), transforms.Lambda(lambda x: x.view(-1))])  
train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)  
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)  
  
# Splitting training data into training and validation sets (80% train, 20% validation).  
train_size = int(0.8 * len(train_dataset))  
val_size = len(train_dataset) - train_size  
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])  
  
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)  
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)  
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Define the train and validation process. Use One-hot encode labels and concatenate as conditional information if we train CVAE and Use dummy zeros for vanilla VAE.

```
In [ ]: def train(model, optimizer, epoch, use_conditional=False, beta=1):
    model.train()
    train_loss = 0
    total_recon_loss = 0
    total_kld_loss = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        if use_conditional:
            # One-hot encode labels and concatenate as conditional information
            c = F.one_hot(target, num_classes=10).float()
        else:
            # Use dummy zeros for non-conditional VAE
            c = torch.zeros(data.size(0), 10)

        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data, c)
        recon_loss = F.binary_cross_entropy(recon_batch, data, reduction='sum')
        kld_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        loss = recon_loss + beta * kld_loss
        loss.backward()
        train_loss += loss.item()
        total_recon_loss += recon_loss.item()
        total_kld_loss += kld_loss.item()
        optimizer.step()

    if epoch%5==0:
        print(f'Epoch {epoch}, Train Loss: {train_loss / len(train_loader.dataset):.4f}, Recon Loss: {total_recon_loss / len(train_loader.datatotal_kld_loss / len(train_loader.dataset):.4f}, KLD Loss: {total_kld_loss / len(train_loader.dataset):.4f}')


def validate(model, use_conditional=False):
    model.eval()
    val_loss = 0
    total_recon_loss = 0
    total_kld_loss = 0
    with torch.no_grad():
        for data, target in val_loader:
            if use_conditional:
                c = F.one_hot(target, num_classes=10).float()
            else:
                c = torch.zeros(data.size(0), 10)

            recon_batch, mu, logvar = model(data, c)
            recon_loss = F.binary_cross_entropy(recon_batch, data, reduction='sum')
            kld_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
            val_loss += (recon_loss + kld_loss).item()
            total_recon_loss += recon_loss.item()
            total_kld_loss += kld_loss.item()

    val_loss /= len(val_loader.dataset)
    total_recon_loss /= len(val_loader.dataset)
    total_kld_loss /= len(val_loader.dataset)
    print(f'Validation Loss: {val_loss:.4f}, Recon Loss: {total_recon_loss:.4f}, KLD Loss: {total_kld_loss:.4f}')



```

Train the VAE.

```
In [ ]: n_in = 784
n_hid = 400
z_dim = 20
learning_rate = 1e-3
model = VAE(n_in=n_in, n_hid=n_hid, z_dim=z_dim)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
n_epochs = 20
use_conditional = False # Set to False for vanilla VAE, True for conditional VAE

for epoch in range(1, n_epochs + 1):
    train(model, optimizer, epoch, use_conditional=use_conditional)
    if epoch%5==0:
        validate(model, use_conditional=use_conditional)

# Saving the model weights
torch.save(model.state_dict(), 'cvae_fashion_mnist.pth' if use_conditional else 'vae_fashion_mnist.pth')
print("VAE model training complete and weights saved.")
```

Epoch 5, Train Loss: 247.2739, Recon Loss: 232.3511, KLD Loss: 14.9228
Validation Loss: 246.7356, Recon Loss: 231.6768, KLD Loss: 15.0588
Epoch 10, Train Loss: 243.2606, Recon Loss: 227.9328, KLD Loss: 15.3279
Validation Loss: 243.0841, Recon Loss: 227.9738, KLD Loss: 15.1103
Epoch 15, Train Loss: 241.6818, Recon Loss: 226.2114, KLD Loss: 15.4704
Validation Loss: 241.8293, Recon Loss: 226.0744, KLD Loss: 15.7548
Epoch 20, Train Loss: 240.7505, Recon Loss: 225.2116, KLD Loss: 15.5390
Validation Loss: 240.8245, Recon Loss: 225.7334, KLD Loss: 15.0911
VAE model training complete and weights saved.

Train the CVAE.

```
In [ ]: n_in = 784
n_hid = 400
z_dim = 20
learning_rate = 1e-3
model = VAE(n_in=n_in, n_hid=n_hid, z_dim=z_dim)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
n_epochs = 20
use_conditional = True # Set to False for vanilla VAE, True for conditional VAE
```

```

for epoch in range(1, n_epochs + 1):
    train(model, optimizer, epoch, use_conditional=use_conditional)
    if epoch%5==0:
        validate(model, use_conditional=use_conditional)

# Saving the model weights
torch.save(model.state_dict(), 'cvae_fashion_mnist.pth' if use_conditional else 'vae_fashion_mnist.pth')
print("CVAE model training complete and weights saved.")

```

Epoch 5, Train Loss: 244.9166, Recon Loss: 231.7806, KLD Loss: 13.1360
Validation Loss: 244.4941, Recon Loss: 230.7921, KLD Loss: 13.7020
Epoch 10, Train Loss: 240.3897, Recon Loss: 227.2638, KLD Loss: 13.1259
Validation Loss: 240.3725, Recon Loss: 226.5769, KLD Loss: 13.7956
Epoch 15, Train Loss: 238.5209, Recon Loss: 225.4796, KLD Loss: 13.0412
Validation Loss: 239.1864, Recon Loss: 225.4931, KLD Loss: 13.6934
Epoch 20, Train Loss: 237.4462, Recon Loss: 224.4569, KLD Loss: 12.9894
Validation Loss: 237.6079, Recon Loss: 224.2056, KLD Loss: 13.4023
CVAE model training complete and weights saved.

Display original images, reconstructed images, and generate new images

```

In [ ]: def display_images(model,use_conditional=True):
    data, target = next(iter(test_loader))
    if use_conditional:
        c = F.one_hot(target, num_classes=10).float()
    else:
        c = torch.zeros(data.size(0), 10)

    with torch.no_grad():
        recon_batch, _, _ = model(data, c)

    fig, axes = plt.subplots(2, 10, figsize=(15, 3))
    for i in range(10):
        axes[0, i].imshow(data[i].view(28, 28), cmap='gray')
        axes[0, i].axis('off')
        axes[0, i].set_title('Original')

        axes[1, i].imshow(recon_batch[i].view(28, 28), cmap='gray')
        axes[1, i].axis('off')
        axes[1, i].set_title('Reconstruct')
    plt.show()

    z = torch.randn(10, z_dim)
    if use_conditional:
        c = F.one_hot(torch.arange(10), num_classes=10).float()
    else:
        c = torch.zeros(10, 10)

    label_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

    with torch.no_grad():
        generated_images = model.decode(z, c)

    # Plot generated images
    fig, axes = plt.subplots(1, 10, figsize=(15, 3))
    for i in range(10):
        axes[i].imshow(generated_images[i].view(28, 28), cmap='gray')
        axes[i].axis('off')
        label = label_names[i] if use_conditional else 'No Label'
        axes[i].set_title(f'c={label}')
    plt.show()

```

Display the effect of VAE.

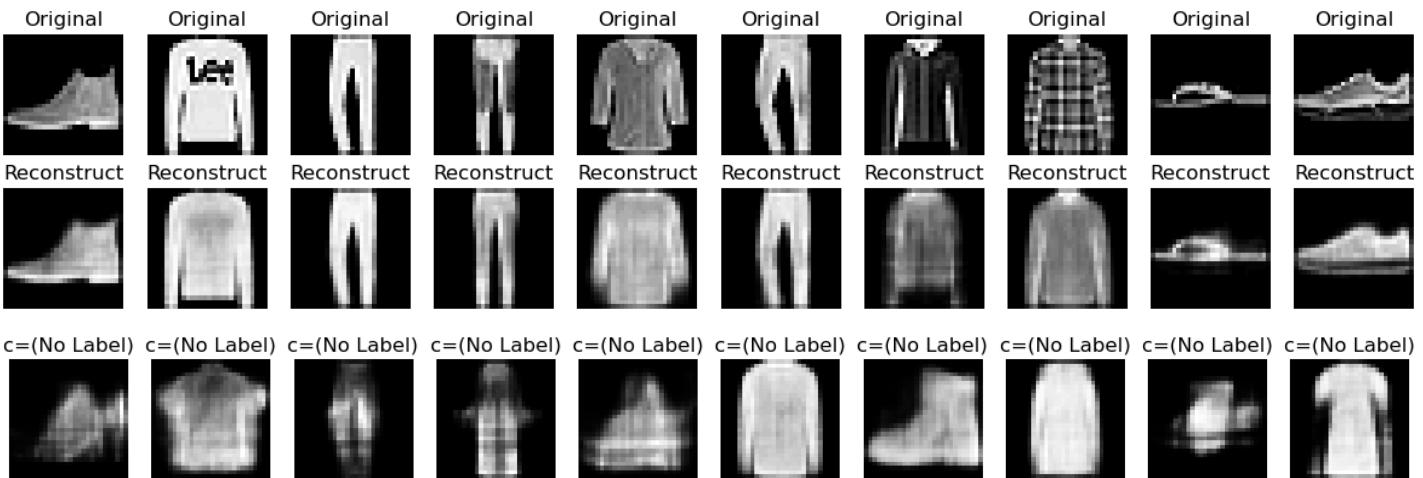
```

In [ ]: use_conditional = False
vae = VAE(n_in=n_in, n_hid=n_hid, z_dim=z_dim)
vae.load_state_dict(torch.load('cvae_fashion_mnist.pth' if use_conditional else 'vae_fashion_mnist.pth'))
vae.eval()

display_images(vae,use_conditional)

```

C:\Users\14857\AppData\Local\Temp\ipykernel_23368\1504968035.py:3: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
vae.load_state_dict(torch.load('cvae_fashion_mnist.pth' if use_conditional else 'vae_fashion_mnist.pth'))

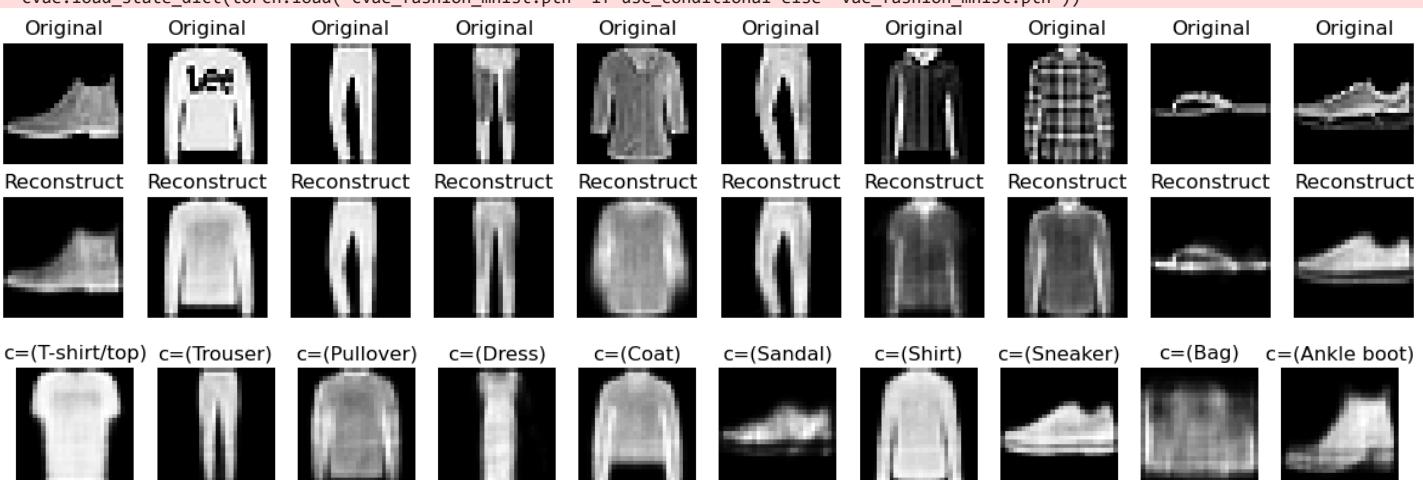


Display the effect of CVAE.

```
In [ ]: use_conditional = True
cvae = VAE(n_in=n_in, n_hid=n_hid, z_dim=z_dim)
cvae.load_state_dict(torch.load('cvae_fashion_mnist.pth' if use_conditional else 'vae_fashion_mnist.pth'))
cvae.eval()

display_images(cvae,use_conditional)
```

C:\Users\14857\AppData\Local\Temp\ipykernel_23368\1840366698.py:3: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.



It can be observed that both VAE and CVAE achieve good reconstruction results. However, in terms of generation performance, CVAE can generate images corresponding to specific classes based on conditional information, and the image quality is higher.

3.3 Manifold Comparison

The visualization of μ .

```
In [ ]: def visualize_latent_space(model, use_conditional=True):
    model.eval()
    all_mu = []
    all_labels = []
    with torch.no_grad():
        for data, target in test_loader:
            if use_conditional:
                c = F.one_hot(target, num_classes=10).float()
            else:
                c = torch.zeros(data.size(0), 10)

            _, mu, _ = model(data, c)
            all_mu.append(mu)
            all_labels.append(target)

    all_mu = torch.cat(all_mu, dim=0).numpy()
    all_labels = torch.cat(all_labels, dim=0).numpy()

    tsne = TSNE(n_components=2, random_state=42)
    latent_2d = tsne.fit_transform(all_mu)
```

```

return latent_2d, all_labels

# Get Latent spaces for VAE and CVAE
vae_latent, vae_labels = visualize_latent_space(vae, use_conditional=False)
cvae_latent, cvae_labels = visualize_latent_space(cvae, use_conditional=True)

fig, axs = plt.subplots(1, 2, figsize=(22, 10))

# Plot for VAE
scatter_vae = axs[0].scatter(vae_latent[:, 0], vae_latent[:, 1], c=vae_labels, cmap='tab10', alpha=0.7)
axs[0].set_title('VAE Latent Space (t-SNE)')
axs[0].set_xlabel('t-SNE Dimension 1')
axs[0].set_ylabel('t-SNE Dimension 2')

# Plot for CVAE
scatter_cvae = axs[1].scatter(cvae_latent[:, 0], cvae_latent[:, 1], c=cvae_labels, cmap='tab10', alpha=0.7)
axs[1].set_title('CVAE Latent Space (t-SNE)')
axs[1].set_xlabel('t-SNE Dimension 1')
axs[1].set_ylabel('t-SNE Dimension 2')

fig.suptitle('Latent Representation of  $\mu$  in VAE and CVAE', fontsize=20)

cbar_ax = fig.add_axes([0.92, 0.15, 0.02, 0.7])
cbar = fig.colorbar(scatter_vae, cax=cbar_ax)
cbar.set_label('Class')

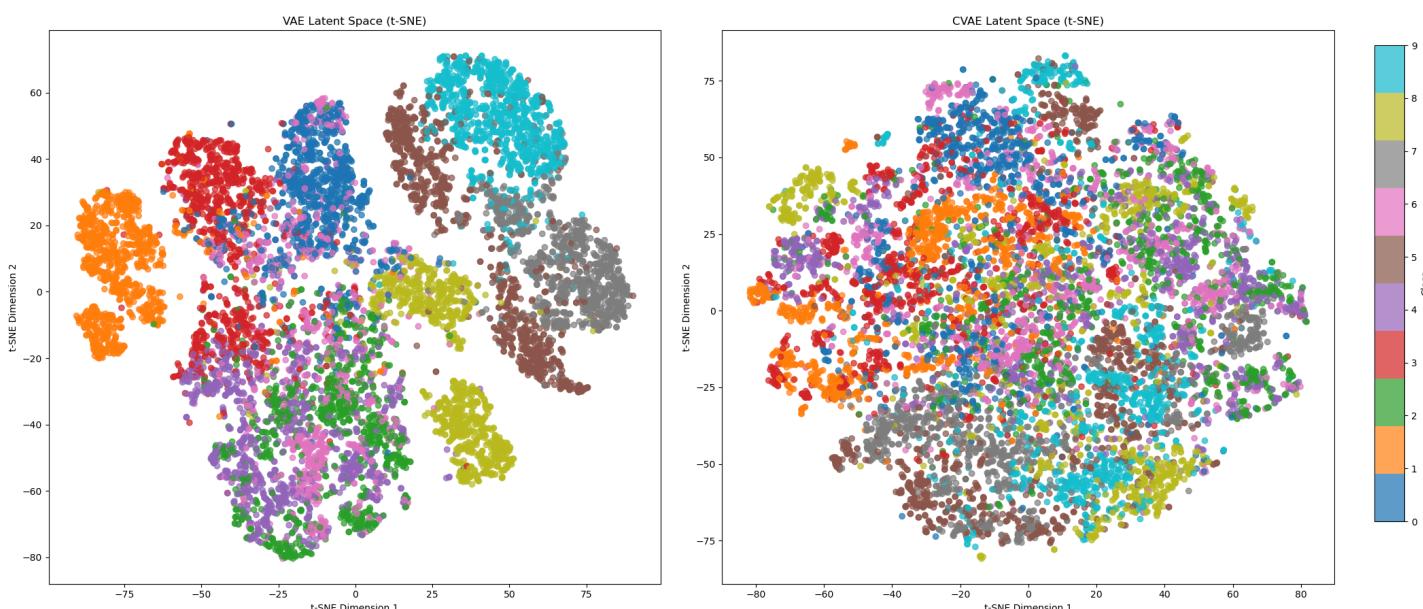
plt.tight_layout(rect=[0, 0, 0.9, 0.95])
plt.subplots_adjust(wspace=0.1)
plt.show()

```

C:\Users\14857\AppData\Local\Temp\ipykernel_23368\3432734237.py:56: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

```
plt.tight_layout(rect=[0, 0, 0.9, 0.95]) # Adjust layout to make room for the colorbar and suptitle
```

Latent Representation of μ in VAE and CVAE



The visualization of σ .

```
In [ ]: def visualize_latent_space(model, use_conditional=True):
    model.eval()
    all_sigma = []
    all_labels = []
    with torch.no_grad():
        for data, target in test_loader:
            if use_conditional:
                c = F.one_hot(target, num_classes=10).float()
            else:
                c = torch.zeros(data.size(0), 10)

            _, _, sigma = model(data, c)
            all_sigma.append(sigma)
            all_labels.append(target)

    all_sigma = torch.cat(all_sigma, dim=0).numpy()
    all_labels = torch.cat(all_labels, dim=0).numpy()

    tsne = TSNE(n_components=2, random_state=42)
    latent_2d = tsne.fit_transform(all_sigma)

    return latent_2d, all_labels

# Get Latent spaces for VAE and CVAE
vae_latent, vae_labels = visualize_latent_space(vae, use_conditional=False)
cvae_latent, cvae_labels = visualize_latent_space(cvae, use_conditional=True)

fig, axs = plt.subplots(1, 2, figsize=(22, 10))

# Plot for VAE
```

```

scatter_vae = axs[0].scatter(vae_latent[:, 0], vae_latent[:, 1], c=vae_labels, cmap='tab10', alpha=0.7)
axs[0].set_title('VAE Latent Space (t-SNE)')
axs[0].set_xlabel('t-SNE Dimension 1')
axs[0].set_ylabel('t-SNE Dimension 2')
# Plot for CVAE
scatter_cvae = axs[1].scatter(cvae_latent[:, 0], cvae_latent[:, 1], c=cvae_labels, cmap='tab10', alpha=0.7)
axs[1].set_title('CVAE Latent Space (t-SNE)')
axs[1].set_xlabel('t-SNE Dimension 1')
axs[1].set_ylabel('t-SNE Dimension 2')

fig.suptitle('Latent Representation of  $\sigma$  in VAE and CVAE', fontsize=20)

cbar_ax = fig.add_axes([0.92, 0.15, 0.02, 0.7])
cbar = fig.colorbar(scatter_vae, cax=cbar_ax)
cbar.set_label('Class')

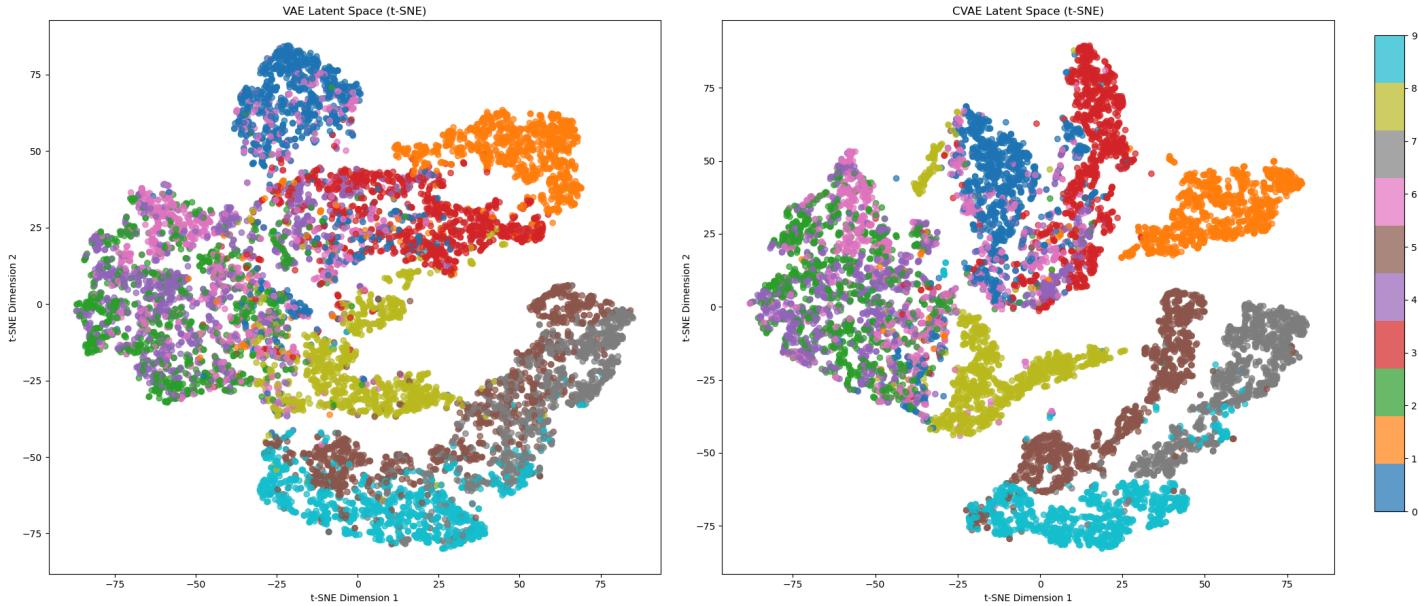
plt.tight_layout(rect=[0, 0, 0.9, 0.95])
plt.subplots_adjust(wspace=0.1)
plt.show()

```

C:\Users\14857\AppData\Local\Temp\ipykernel_23368\2402147482.py:56: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

plt.tight_layout(rect=[0, 0, 0.9, 0.95]) # Adjust layout to make room for the colorbar and suptitle

Latent Representation of σ in VAE and CVAE



When visualizing the latent representations learned by both the VAE and CVAE using t-SNE, the following observations were made: in the latent space of μ , the VAE's latent representation shows a clearer separation between classes, while the CVAE's representation of the latent variable μ appears more mixed across classes. However, the variance σ for the CVAE is more distinct and well-defined compared to that in the VAE.

Hypothesis: This phenomenon occurs because in the CVAE, the class labels are used as conditional information during training. As a result, the need for latent variables to capture the class-related differences is reduced, since the class is already explicitly provided to the model. This leads to a more mixed distribution for μ across classes in the CVAE. On the other hand, since the class differences are explicitly conditioned, the latent variable σ , which represents the variance, can better capture subtle variations within each class, resulting in a clearer representation. This highlights the effect of conditioning in disentangling different factors of variation—specifically, by focusing the model on class-internal differences rather than class distinctions.

时间序列分析代码作业

姓名：吕欣远

学号：2000014921

第一题：股票投资

第一题股票结果报告

Xinyuan Lyu

2024-06-04

```
knitr::opts_chunk$set(echo = TRUE, cache = TRUE)
options(warn=-1)

options(warn=-1)

library("readxl")
library("zoo")

## 
## 载入程辑包: 'zoo'

## The following objects are masked from 'package:base':
## 
##     as.Date, as.Date.numeric

library("xts")
library(urca)
library(stargazer)

## 
## Please cite as:

## Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.

## R package version 5.2.3. https://CRAN.R-project.org/package=stargazer

library(xtable)
library(forecast)

## Registered S3 method overwritten by 'quantmod':
##   method           from
##   as.zoo.data.frame zoo

library(dplyr)
```

Part A

```
data <- read_excel("C:/Users/14857/Desktop/时序分析/final project(1)/final project/HS300 目.xlsx",sheet=1)

## New names:
## * ` ` -> `...1`
## * `603195.SH` -> `603195.SH'...142`
## * `603195.SH` -> `603195.SH'...301`

data <- na.omit(data)
colnames(data)[1] <- "date"
data$date <- as.Date(as.character(data$date), format="%Y%m%d")
data_ts <- xts(data[,-which(names(data) %in% "date")], order.by=data$date)
data

## # A tibble: 2,429 x 301
##   date      `000977.SZ` `000768.SZ` `002466.SZ` `000338.SZ` `000157.SZ` ...
##   <date>     <chr>     <chr>     <chr>     <chr>     <chr>
## 1 2013-01-04 0.85    -1.08    -0.28     0.4      -0.11
## 2 2013-01-07 0.91    5.1      -2.83    -1.42    -0.11
## 3 2013-01-08 4.37    0.46     0.03     -1.6     NaN
## 4 2013-01-09 1       -2.19    0.61     -0.41    -3.16
## 5 2013-01-10 -1.58   -0.12    0.16     2.24     0
## 6 2013-01-11 -1.8    3.3      -3.94    -4.78    -2.92
## 7 2013-01-14 3.4     10.03   2.95     3.35     3.94
## 8 2013-01-15 2.9     4.15     2.54    -1.21    1.56
## 9 2013-01-16 -0.51   6.37     0.38     0.53    -0.99
## 10 2013-01-17 1.03   -7.2     -3.69    -2       -0.78
## # i 2,419 more rows
## # i 295 more variables: `002415.SZ` <chr>, `600085.SH` <chr>,
## # `002049.SZ` <chr>, `600011.SH` <chr>, `002841.SZ` <chr>,
## # `002736.SZ` <chr>, `300782.SZ` <chr>, `002304.SZ` <chr>,
## # `002812.SZ` <chr>, `300628.SZ` <chr>, `002252.SZ` <chr>,
## # `002001.SZ` <chr>, `600019.SH` <chr>, `000938.SZ` <chr>,
## # `002050.SZ` <chr>, `002555.SZ` <chr>, `300059.SZ` <chr>, ...
```

进行无趋势的 ADF 测试，看平稳性。

```
results_without_trend_df <- data.frame(Names = character(),tau = numeric(),critical_value = numeric(),trend = character())

for (i in 1:ncol(data_ts)) {
  stock_name <- colnames(data_ts)[i]
  series <- data_ts[, i]
  series <- as.numeric(series)
  series <- na.omit(series)
  mode <- "drift"
  test <- ur.df(series, type = mode)
  test_summary <- summary(test)

  tau_value <- test_summary@teststat[1]
  critical_value <- test_summary@cval[1]
  has_trend <- "No"
  is_stationary <- if (tau_value<critical_value) "Yes" else "No"
```

```

new_row <- data.frame(Names = stock_name, tau = tau_value, critical_value = critical_value, trend = trend)

results_without_trend_df <- rbind(results_without_trend_df, new_row) # 将新行添加到结果表格中
}
results_without_trend_df

```

	Names	tau	critical_value	trend	stationary
## 1	000977.SZ'	-33.91902	-3.43	No	Yes
## 2	000768.SZ'	-33.49281	-3.43	No	Yes
## 3	002466.SZ'	-32.09402	-3.43	No	Yes
## 4	000338.SZ'	-35.34389	-3.43	No	Yes
## 5	000157.SZ'	-34.75736	-3.43	No	Yes
## 6	002415.SZ'	-35.57920	-3.43	No	Yes
## 7	600085.SH'	-36.12507	-3.43	No	Yes
## 8	002049.SZ'	-30.80656	-3.43	No	Yes
## 9	600011.SH'	-34.10577	-3.43	No	Yes
## 10	002841.SZ'	-24.40612	-3.43	No	Yes
## 11	002736.SZ'	-29.15893	-3.43	No	Yes
## 12	300782.SZ'	-19.02927	-3.43	No	Yes
## 13	002304.SZ'	-36.60648	-3.43	No	Yes
## 14	002812.SZ'	-25.60243	-3.43	No	Yes
## 15	300628.SZ'	-25.82560	-3.43	No	Yes
## 16	002252.SZ'	-27.62586	-3.43	No	Yes
## 17	002001.SZ'	-35.80175	-3.43	No	Yes
## 18	600019.SH'	-36.56768	-3.43	No	Yes
## 19	000938.SZ'	-30.25821	-3.43	No	Yes
## 20	002050.SZ'	-34.47308	-3.43	No	Yes
## 21	002555.SZ'	-30.07448	-3.43	No	Yes
## 22	300059.SZ'	-33.26575	-3.43	No	Yes
## 23	000001.SZ'	-34.93901	-3.43	No	Yes
## 24	300498.SZ'	-41.26970	-3.43	No	Yes
## 25	000166.SZ'	-31.69981	-3.43	No	Yes
## 26	002064.SZ'	-36.05800	-3.43	No	Yes
## 27	300450.SZ'	-26.44898	-3.43	No	Yes
## 28	600009.SH'	-37.16492	-3.43	No	Yes
## 29	600036.SH'	-36.82048	-3.43	No	Yes
## 30	003816.SZ'	-21.87770	-3.43	No	Yes
## 31	300601.SZ'	-23.34767	-3.43	No	Yes
## 32	300316.SZ'	-33.24573	-3.43	No	Yes
## 33	002600.SZ'	-30.09502	-3.43	No	Yes
## 34	300347.SZ'	-35.44124	-3.43	No	Yes
## 35	300142.SZ'	-32.50829	-3.43	No	Yes
## 36	600028.SH'	-35.55109	-3.43	No	Yes
## 37	600000.SH'	-35.49557	-3.43	No	Yes
## 38	002129.SZ'	-31.13065	-3.43	No	Yes
## 39	300033.SZ'	-32.81771	-3.43	No	Yes
## 40	600016.SH'	-35.74731	-3.43	No	Yes
## 41	002120.SZ'	-32.84683	-3.43	No	Yes
## 42	000333.SZ'	-35.53012	-3.43	No	Yes
## 43	000538.SZ'	-35.65871	-3.43	No	Yes
## 44	000425.SZ'	-36.23598	-3.43	No	Yes
## 45	000708.SZ'	-33.75966	-3.43	No	Yes
## 46	300760.SZ'	-25.20138	-3.43	No	Yes

```

## 263      688169.SH' -20.26156      -3.43    No     Yes
## 264      600183.SH' -36.15317      -3.43    No     Yes
## 265      601808.SH' -36.16720      -3.43    No     Yes
## 266      600886.SH' -35.01647      -3.43    No     Yes
## 267      601633.SH' -34.03324      -3.43    No     Yes
## 268      600845.SH' -34.79891      -3.43    No     Yes
## 269      601669.SH' -32.17319      -3.43    No     Yes
## 270      601799.SH' -34.35668      -3.43    No     Yes
## 271      601898.SH' -34.97298      -3.43    No     Yes
## 272      601021.SH' -30.20501      -3.43    No     Yes
## 273      601328.SH' -34.84779      -3.43    No     Yes
## 274      688036.SH' -21.60728      -3.43    No     Yes
## 275      600809.SH' -35.35533      -3.43    No     Yes
## 276      688008.SH' -26.60627      -3.43    No     Yes
## 277      600547.SH' -32.93464      -3.43    No     Yes
## 278      603233.SH' -27.47943      -3.43    No     Yes
## 279      603882.SH' -22.57385      -3.43    No     Yes
## 280      600905.SH' -13.04969      -3.44    No     Yes
## 281      601186.SH' -34.52133      -3.43    No     Yes
## 282      601225.SH' -33.66213      -3.43    No     Yes
## 283      601288.SH' -38.57289      -3.43    No     Yes
## 284      601919.SH' -32.46348      -3.43    No     Yes
## 285      600438.SH' -31.79265      -3.43    No     Yes
## 286      600999.SH' -33.97094      -3.43    No     Yes
## 287      600362.SH' -34.81359      -3.43    No     Yes
## 288      601238.SH' -35.25334      -3.43    No     Yes
## 289      601868.SH' -16.49330      -3.44    No     Yes
## 290      601766.SH' -30.83885      -3.43    No     Yes
## 291      601800.SH' -32.71775      -3.43    No     Yes
## 292      601618.SH' -33.41375      -3.43    No     Yes
## 293      601162.SH' -19.39009      -3.43    No     Yes
## 294      603501.SH' -23.82289      -3.43    No     Yes
## 295      601939.SH' -37.65038      -3.43    No     Yes
## 296      601933.SH' -35.57120      -3.43    No     Yes
## 297      600837.SH' -35.25520      -3.43    No     Yes
## 298      601155.SH' -35.01592      -3.43    No     Yes
## 299      600745.SH' -27.80277      -3.43    No     Yes
## 300 603195.SH'...301 -17.24748      -3.43    No     Yes

```

我们发现所有股票序列都是平稳的。

进行可以选择是否有趋势的 ADF 测试。

```

results_df <- data.frame(Names = character(), tau = numeric(), critical_value = numeric(), trend = character())

for (i in 1:ncol(data_ts)) {
  stock_name <- colnames(data_ts)[i]
  series <- data_ts[, i]
  series <- as.numeric(series)
  series <- na.omit(series)
  # 创建时间变量
  time <- 1:length(series)
  fit <- lm(series ~ time)
  # 获取时间项的 p-值
  results_df[i, ] <- c(stock_name, tau = summary(fit)$coefficients[2, 1], critical_value = qnorm(0.95), trend = ifelse(sign(summary(fit)$coefficients[2, 1]) == 0, "No", "Yes"))
}

```

```

p_value <- summary(fit)$coefficients[2, 4]
# 根据 p-值选择模型是否有 trend
mode <- if (p_value < 0.05) "trend" else "drift"
test <- ur.df(series, type = mode)
test_summary <- summary(test)

tau_value <- test_summary@teststat[1]
critical_value <- test_summary@cval[1]
has_trend <- if (mode == "trend") "Yes" else "No"
is_stationary <- if (tau_value < critical_value) "Yes" else "No"

new_row <- data.frame(Names = stock_name, tau = tau_value, critical_value = critical_value, trend =
# 将新行添加到结果表格中
results_df <- rbind(results_df, new_row)
}
results_df

```

	Names	tau	critical_value	trend	stationary
## 1	000977.SZ'	-34.00432	-3.96	Yes	Yes
## 2	000768.SZ'	-33.49281	-3.43	No	Yes
## 3	002466.SZ'	-32.09402	-3.43	No	Yes
## 4	000338.SZ'	-35.34389	-3.43	No	Yes
## 5	000157.SZ'	-34.75736	-3.43	No	Yes
## 6	002415.SZ'	-35.57920	-3.43	No	Yes
## 7	600085.SH'	-36.12507	-3.43	No	Yes
## 8	002049.SZ'	-30.80656	-3.43	No	Yes
## 9	600011.SH'	-34.10577	-3.43	No	Yes
## 10	002841.SZ'	-24.53399	-3.96	Yes	Yes
## 11	002736.SZ'	-29.17905	-3.96	Yes	Yes
## 12	300782.SZ'	-19.46105	-3.96	Yes	Yes
## 13	002304.SZ'	-36.60648	-3.43	No	Yes
## 14	002812.SZ'	-25.63544	-3.96	Yes	Yes
## 15	300628.SZ'	-25.88460	-3.96	Yes	Yes
## 16	002252.SZ'	-27.83143	-3.96	Yes	Yes
## 17	002001.SZ'	-35.80175	-3.43	No	Yes
## 18	600019.SH'	-36.56768	-3.43	No	Yes
## 19	000938.SZ'	-30.31962	-3.96	Yes	Yes
## 20	002050.SZ'	-34.47308	-3.43	No	Yes
## 21	002555.SZ'	-30.15582	-3.96	Yes	Yes
## 22	300059.SZ'	-33.35567	-3.96	Yes	Yes
## 23	000001.SZ'	-34.93901	-3.43	No	Yes
## 24	300498.SZ'	-41.26970	-3.43	No	Yes
## 25	000166.SZ'	-31.69981	-3.43	No	Yes
## 26	002064.SZ'	-36.05800	-3.43	No	Yes
## 27	300450.SZ'	-26.57499	-3.96	Yes	Yes
## 28	600009.SH'	-37.16492	-3.43	No	Yes
## 29	600036.SH'	-36.82048	-3.43	No	Yes
## 30	003816.SZ'	-21.87770	-3.43	No	Yes
## 31	300601.SZ'	-23.63853	-3.96	Yes	Yes
## 32	300316.SZ'	-33.24573	-3.43	No	Yes
## 33	002600.SZ'	-30.09502	-3.43	No	Yes
## 34	300347.SZ'	-35.44124	-3.43	No	Yes

## 251	601901.SH'	-32.30507	-3.43	No	Yes
## 252	600352.SH'	-33.42951	-3.43	No	Yes
## 253	600406.SH'	-35.12693	-3.43	No	Yes
## 254	601111.SH'	-34.26354	-3.43	No	Yes
## 255	600332.SH'	-37.02143	-3.43	No	Yes
## 256	603799.SH'	-28.78565	-3.96	Yes	Yes
## 257	601601.SH'	-36.15162	-3.43	No	Yes
## 258	600887.SH'	-36.89391	-3.43	No	Yes
## 259	601006.SH'	-37.97546	-3.43	No	Yes
## 260	600436.SH'	-35.69561	-3.43	No	Yes
## 261	603993.SH'	-35.20167	-3.43	No	Yes
## 262	601211.SH'	-33.53923	-3.43	No	Yes
## 263	688169.SH'	-20.35604	-3.96	Yes	Yes
## 264	600183.SH'	-36.15317	-3.43	No	Yes
## 265	601808.SH'	-36.16720	-3.43	No	Yes
## 266	600886.SH'	-35.01647	-3.43	No	Yes
## 267	601633.SH'	-34.03324	-3.43	No	Yes
## 268	600845.SH'	-34.79891	-3.43	No	Yes
## 269	601669.SH'	-32.17319	-3.43	No	Yes
## 270	601799.SH'	-34.35668	-3.43	No	Yes
## 271	601898.SH'	-34.97298	-3.43	No	Yes
## 272	601021.SH'	-30.22767	-3.96	Yes	Yes
## 273	601328.SH'	-34.84779	-3.43	No	Yes
## 274	688036.SH'	-21.68855	-3.96	Yes	Yes
## 275	600809.SH'	-35.35533	-3.43	No	Yes
## 276	688008.SH'	-26.60627	-3.43	No	Yes
## 277	600547.SH'	-32.93464	-3.43	No	Yes
## 278	603233.SH'	-27.47943	-3.43	No	Yes
## 279	603882.SH'	-22.63593	-3.96	Yes	Yes
## 280	600905.SH'	-13.06266	-3.98	Yes	Yes
## 281	601186.SH'	-34.52133	-3.43	No	Yes
## 282	601225.SH'	-33.66213	-3.43	No	Yes
## 283	601288.SH'	-38.57289	-3.43	No	Yes
## 284	601919.SH'	-32.46348	-3.43	No	Yes
## 285	600438.SH'	-31.79265	-3.43	No	Yes
## 286	600999.SH'	-33.97094	-3.43	No	Yes
## 287	600362.SH'	-34.81359	-3.43	No	Yes
## 288	601238.SH'	-35.25334	-3.43	No	Yes
## 289	601868.SH'	-16.49330	-3.44	No	Yes
## 290	601766.SH'	-30.83885	-3.43	No	Yes
## 291	601800.SH'	-32.71775	-3.43	No	Yes
## 292	601618.SH'	-33.41375	-3.43	No	Yes
## 293	601162.SH'	-19.50438	-3.96	Yes	Yes
## 294	603501.SH'	-24.09189	-3.96	Yes	Yes
## 295	601939.SH'	-37.65038	-3.43	No	Yes
## 296	601933.SH'	-35.57120	-3.43	No	Yes
## 297	600837.SH'	-35.25520	-3.43	No	Yes
## 298	601155.SH'	-35.01592	-3.43	No	Yes
## 299	600745.SH'	-27.80277	-3.43	No	Yes
## 300	603195.SH'...301	-17.29610	-3.96	Yes	Yes

结论没有变化，所有序列都是平稳的。

Part B

Part B

我们先分割数据，分为训练集和测试集。

```
# 分割训练集和测试集
train_data <- data %>% filter(date <= as.Date("2022-11-30"))
test_data <- data %>% filter(date > as.Date("2022-11-30"))
# 转化为 ts 数据
train_data <- xts(train_data[,-which(names(train_data) %in% "date")], order.by=train_data$date)
test_data <- xts(test_data[,-which(names(test_data) %in% "date")], order.by=test_data$date)
```

进行初始的拟合，选择 auto.arima 并且设置 d=0.

```
# 创建一个列表来存储每个股票的模型
models <- list()

# 对每个股票数据进行拟合
for(stock in colnames(train_data)) {
  stock_data <- train_data[, stock]

  # 将数据转换为数值型
  numeric_stock_data <- as.numeric(as.character(stock_data))

  # 创建一个 xts 对象，将数值型数据和日期结合
  numeric_stock_data_xts <- xts(numeric_stock_data, order.by = index(train_data))

  # 确保数据没有缺失值
  numeric_stock_data_xts <- na.omit(numeric_stock_data_xts)

  # 对股票数据进行 auto.arima 拟合
  models[[stock]] <- auto.arima(numeric_stock_data_xts, d=0)
}
```

我们接下来进一步迭代预测。

这里的逻辑在于此：即我们不能用同一个模型一次性预测后面一个月的所有数据。因为后面的预测值会不断加大误差，导致预测愈来愈不准确。因此我们的方法是，即使要做投资，每天结束也会知道当天的真实数据，因此我们每一天结束更新模型，将当天的实际数据作为信息给模型。因此每天只预测明天一天的数据，并做出投资决定。

下面给出我们对于未来一个月的每只股票收益率的预测值，每次预测基于前一天。

```
# 初始化存储预测结果的数据框
predictions <- data.frame(matrix(ncol = ncol(train_data), nrow = 21))
colnames(predictions) <- colnames(train_data)

# 确保 actual_data 是数值型矩阵
actual_data <- apply(as.matrix(test_data), 2, as.numeric)

# 对每一天进行预测
for (day in 1:21) {
  # 存储每只股票的当天预测值
  day_predictions <- numeric(ncol(train_data))

  for (i in 1:ncol(train_data)) {
    stock <- colnames(train_data)[i]
```

```

# 使用模型进行一天的预测
forecast_result <- forecast(models[[stock]], h=1)
day_predictions[i] <- forecast_result$mean[1]

# 更新模型的训练数据，加入新一天的实际收益率
new_data <- c(models[[stock]]$x, actual_data[day, i])
# 重新拟合模型
models[[stock]] <- auto.arima(new_data, d=0)
}

# 将当天的预测值存储到 predictions 数据框中
predictions[day, ] <- day_predictions
}

# 查看预测结果
print(predictions)

##    000977.SZ'  000768.SZ'  002466.SZ'  000338.SZ'  000157.SZ'  002415.SZ'
## 1  0.12855195 -0.039123247  0.42905847      0 -0.19349762 -0.065293565
## 2  0.28740101  0.110573866  0.10555241      0 -0.10426467  0.150035807
## 3  0.13481368 -0.225010126  0.02961944      0  0.18087789  0.035006096
## 4  0.19690244  0.317263925  0.17126839      0  0.11546097  0.193162357
## 5  0.10421696  0.081398972  0.27691605      0 -0.23930348 -0.035559539
## 6  0.10400740  0.020420340  0.21239622      0 -0.10874626  0.013816783
## 7  0.07396571 -0.232326071  0.09264854      0  0.23433793  0.140461463
## 8  0.10813703  0.154731845  0.16837554      0  0.10726686  0.161473859
## 9  0.18115295  0.302440814 -0.06089359      0 -0.24308904  0.005560751
## 10 0.02898117  0.121886552  0.05779764      0 -0.04232851  0.137430992
## 11 0.14797775  0.120545989  0.16952884      0  0.21307185  0.101363727
## 12 0.15742938  0.240630593  0.35892253      0  0.05949838  0.064914260
## 13 0.06458412 -0.038952406 -0.27741308      0 -0.15110810  0.092192179
## 14 0.05419785  0.006088352  0.67860752      0 -0.08142490  0.166329987
## 15 0.06351138  0.127400089 -0.26253280      0  0.21024366  0.088550811
## 16 0.09289605  0.130094308  0.03657326      0  0.11379052  0.203471596
## 17 0.11580823 -0.141791504 -0.10494483      0 -0.17748151  0.100629845
## 18 0.18538998  0.096834453  0.20790170      0 -0.07959344  0.100308579
## 19 0.18188377  0.517520710  0.35824558      0  0.16548499  0.051592509
## 20 0.12588675  0.269955933  0.16360348      0  0.05730411  0.057654046
## 21 0.03886649 -0.131084856  0.01490843      0 -0.16094104  0.130226796
##    600085.SH'  002049.SZ'  600011.SH'  002841.SZ'  002736.SZ'  300782.SZ'
## 1   0.347690276  0.055351347 -0.07913427 -0.27577441  0.001541237  0.323384367
## 2  -0.250268973  0.445129401 -0.05724837 -0.21301951  0.040152960  0.799906248
## 3  -0.130452468  0.349712563  0.16818702 -0.15752490  0.058682676  0.384869881
## 4   0.383074964  0.092990144  0.14186136 -0.14751449  0.313826459  0.207782713
## 5   0.162181972  0.182079145 -0.37668509  0.20855913  0.202480551  0.480210383
## 6  -0.010725051  0.224104133 -0.04171748  0.12319519  0.109848109  0.004788748
## 7  -0.241489942  0.084812107  0.32588292 -0.09954637  0.130026489 -0.044053847
## 8   0.198817073  0.129967534 -0.08144524 -0.14884988  0.169851419  0.176244929
## 9   0.341149973  0.242431257 -0.35359137 -0.26772256 -0.103292420  0.350273815
## 10 -0.086275364 -0.241569478  0.07020374 -0.28553081 -0.005519597 -0.014499210
## 11 -0.126685004 -0.040284818  0.21408832 -0.17585044  0.012897242  0.119679070
## 12  0.131885087  0.169711447  0.03507358 -0.29456955 -0.142734859  0.120871448
## 13  0.100855492  0.163328101 -0.19129679 -0.31943612 -0.054129847 -0.042754320

```

```

## 16      -1.09580561
## 17      -0.48520371
## 18       0.03251508
## 19      -0.07030631
## 20      -0.05855554
## 21      0.23798482

```

这里展示一下该月份的真实值.

```

# 初始化存储实际收益率的数据框
actual_returns <- data.frame(matrix(ncol = ncol(train_data), nrow = 21))
colnames(actual_returns) <- colnames(train_data)

# 将 actual_data 转换为 actual_returns 数据框
for (day in 1:21) {
  for (i in 1:ncol(train_data)) {
    stock <- colnames(train_data)[i]
    actual_returns[day, stock] <- actual_data[day, i]
  }
}

# 查看生成的 actual_returns 数据框
print(actual_returns)

```

```

##   000977.SZ' 000768.SZ' 002466.SZ' 000338.SZ' 000157.SZ' 002415.SZ' 600085.SH'
## 1      3.1982     0.7100    -0.4105     0.4660    -0.5137     1.7137     0.0000
## 2      0.1746    -1.1874    -1.4817    -2.0408    -0.3442    -0.3120    -1.2614
## 3      1.3072     1.8025    -0.1470     2.9356     2.9361     4.6322     0.7418
## 4     -0.5591     1.0328     1.0419     0.7360    -0.5034     2.0640    -1.5750
## 5     -0.6488    -1.6064     0.5044    -1.1872    -0.8432    -0.9379     0.8105
## 6     -1.2625    -1.2245    -0.7472     0.1848     0.1701    -0.2959     0.0618
## 7     -0.5732    -1.2397    -0.1124     0.6458     1.3582     2.4926     0.7623
## 8      0.9756     0.2282    -2.3060    -1.3749    -2.5126    -0.4343     2.1672
## 9     -2.1080    -1.8975    -1.3472     0.7435     0.1718     0.6688    -2.2814
## 10     0.2243     0.6190    -0.1167     0.8303    -0.1715     1.0687    -3.9320
## 11     0.5372    -0.0769     1.8696    -1.0064    -1.3746    -0.2286     0.1066
## 12     -1.3802    -1.4621    -4.2556     0.0000     0.6969    -1.6328     1.1499
## 13     -1.6704    -1.2886     4.6484    -2.0333    -2.7682    -0.4077    -3.1579
## 14     -1.4692    -0.3560    -3.8580    -0.7547    -0.8897    -2.1930    -0.8478
## 15     -0.8388    -1.0322    -1.7623    -0.6654    -0.1795     0.6278    -0.7016
## 16     -0.3289    -3.2892    -2.8727     0.1914    -1.2590     1.0101    -2.2522
## 17      1.1315    -1.1613     0.1373    -1.5282    -0.3643     1.8529    -0.9261
## 18      1.1189     2.7277     1.9566    -0.1940     0.9141     0.8374     3.1464
## 19     -0.0461     0.1225     0.1467     0.7775     0.5435    -1.1741     0.0884
## 20     -1.8911    -1.3872    -1.5379    -2.1215    -1.4414    -1.4778    -2.0318
## 21      0.2351     3.9719     0.1983    -0.1970    -0.9141     0.7353     0.7665
##   002049.SZ' 600011.SH' 002841.SZ' 002736.SZ' 300782.SZ' 002304.SZ' 002812.SZ'
## 1      4.3273    -1.5484     0.9294    -0.1095     3.3794     2.8293     4.6214
## 2     -0.1093    -0.7864    -0.3231     0.0000    -0.8403     0.1926     2.4814
## 3     -0.7881     3.4346     0.5997     2.1930    -1.9237     1.2838    -1.9640
## 4      0.7723    -3.3206     2.3039     0.1073     1.6504     2.3046     3.5785
## 5      0.2993    -3.5667    -0.3150    -0.8574    -3.7402    -0.0663     2.6133
## 6     -1.5356    -1.3699    -1.2322    -0.2162    -1.3599     1.1470     0.2158

```

```

## 1      4.0674
## 2     -1.4085
## 3      3.0357
## 4      1.8718
## 5     -1.1228
## 6      0.0344
## 7      1.4517
## 8     -3.0381
## 9      0.7134
## 10     0.3889
## 11     -1.1552
## 12     -0.2799
## 13     -0.5755
## 14     -2.1176
## 15      0.0577
## 16      1.5495
## 17      1.0575
## 18     -1.7557
## 19      1.5012
## 20     -1.1339
## 21     -1.4318

```

下面进行一些投资策略的建构和结果的展示，得到每天对于每一只股票应该投资多少的策略矩阵。策略 1 是按照比例投资正预测收益率的股票，也即题目所给出的投资规则。策略 2 是平等投资每一只正预测收益率的股票，即每天平均投资该天预测收益率为正值的股票。策略 3 是平等投资所有股票，无论正负，也即对于每一只股票投资比例都是 $1/300$ 。

策略 1：按照比例投资正预测收益率的股票

```

# 创建一个空的矩阵来存储策略数据，也即策略矩阵
strategy_1 <- matrix(0, nrow = 21, ncol = 300)

# 遍历每一天的数据行
for (i in 1:21) {
  # 获取当前行的预测收益率
  daily_predictions <- predictions[i, ]

  # 找出正收益率
  positive_returns <- daily_predictions[daily_predictions > 0]

  # 计算正收益率的总和
  sum_positive_returns <- sum(positive_returns)

  # 计算每只股票的投资比例
  if (sum_positive_returns > 0) {
    strategy_1[i, daily_predictions > 0] <- positive_returns / sum_positive_returns
  }
}

# 将 strategy 矩阵转换为数据框
strategy_1 <- as.data.frame(strategy_1)

# 设置列名与 predictions 相同
colnames(strategy_1) <- colnames(predictions)

```

```

# 避免缺失值影响
strategy_1[is.na(strategy_1)] <- 0

print(strategy_1)

##      000977.SZ'    000768.SZ'    002466.SZ'  000338.SZ'    000157.SZ'    002415.SZ'
## 1  0.0034433359  0.0000000000  0.0114925716          0  0.0000000000  0.0000000000
## 2  0.0057929807  0.0022287753  0.0021275607          0  0.0000000000  0.0030241875
## 3  0.0041047853  0.0000000000  0.0009018479          0  0.005507341  0.0010658600
## 4  0.0031871320  0.0051353452  0.0027722103          0  0.001868892  0.0031265937
## 5  0.0023115009  0.0018054047  0.0061419148          0  0.0000000000  0.0000000000
## 6  0.0023107001  0.0004536724  0.0047187411          0  0.0000000000  0.0003069632
## 7  0.0018472498  0.0000000000  0.0023138424          0  0.005852451  0.0035079418
## 8  0.0024247837  0.0034695909  0.0037755268          0  0.002405272  0.0036207688
## 9  0.0059009167  0.0098517744  0.0000000000          0  0.0000000000  0.0001811371
## 10 0.0007140262  0.0030029912  0.0014239947          0  0.0000000000  0.0033859688
## 11 0.0037531544  0.0030574036  0.0042997539          0  0.005404134  0.0025708846
## 12 0.0042649800  0.0065190162  0.0097237086          0  0.001611893  0.0017586173
## 13 0.0023769517  0.0000000000  0.0000000000          0  0.0000000000  0.0033930378
## 14 0.0022444362  0.0002521303  0.0281024290          0  0.0000000000  0.0068880414
## 15 0.0022071693  0.0044274519  0.0000000000          0  0.007306460  0.0030773484
## 16 0.0024924016  0.0034904309  0.0009812607          0  0.003053000  0.0054591439
## 17 0.0036535763  0.0000000000  0.0000000000          0  0.0000000000  0.0031747210
## 18 0.0070620430  0.0036887057  0.0079195799          0  0.0000000000  0.0038210452
## 19 0.0042905761  0.0122081373  0.0084508912          0  0.003903734  0.0012170497
## 20 0.0028579603  0.0061287099  0.0037142293          0  0.001300954  0.0013088985
## 21 0.0014509756  0.0000000000  0.0005565660          0  0.0000000000  0.0048616658
##      600085.SH'    002049.SZ'    600011.SH'    002841.SZ'    002736.SZ'    300782.SZ'
## 1  0.009313079  0.0014826169  0.0000000000  0.0000000000  4.128289e-05  0.0086620316
## 2  0.0000000000  0.0089722233  0.0000000000  0.0000000000  8.093407e-04  0.0161232610
## 3  0.0000000000  0.0106479924  0.0051209317  0.0000000000  1.786761e-03  0.0117184569
## 4  0.006200586  0.0015051711  0.0022962178  0.0000000000  5.079705e-03  0.0033632439
## 5  0.003597147  0.0040384607  0.0000000000  0.0046257790  4.490958e-03  0.0106509220
## 6  0.0000000000  0.0049788521  0.0000000000  0.0027369893  2.440461e-03  0.0001063901
## 7  0.0000000000  0.0021181321  0.0081387328  0.0000000000  3.247334e-03  0.0000000000
## 8  0.004458125  0.0029142946  0.0000000000  0.0000000000  3.808621e-03  0.0039519842
## 9  0.011112695  0.0078970098  0.0000000000  0.0000000000  0.000000e+00  0.0114098973
## 10 0.0000000000  0.0000000000  0.0017296511  0.0000000000  0.000000e+00  0.0000000000
## 11 0.0000000000  0.0000000000  0.0054299144  0.0000000000  3.271123e-04  0.0030354160
## 12 0.003572950  0.0045977183  0.0009501919  0.0000000000  0.000000e+00  0.0032745750
## 13 0.003711882  0.0060111218  0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## 14 0.0000000000  0.0007207681  0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## 15 0.005422435  0.0008393075  0.0097093208  0.0000000000  0.000000e+00  0.0124760938
## 16 0.004832732  0.0000000000  0.0031719605  0.0000000000  0.000000e+00  0.0026260503
## 17 0.0000000000  0.0000000000  0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## 18 0.0000000000  0.0010073019  0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## 19 0.003701291  0.0057824084  0.0074158236  0.0000000000  0.000000e+00  0.0141045291
## 20 0.0000000000  0.0154505326  0.0000000000  0.0000000000  0.000000e+00  0.0000000000
## 21 0.002707523  0.0162901108  0.0017625456  0.0006784969  0.000000e+00  0.0076076459
##      002304.SZ'    002812.SZ'  300628.SZ'    002252.SZ'    002001.SZ'    600019.SH'
## 1  0.0000000000  0.0000000000          0  0.0043241389  0.002684418  0.0000000000
## 2  0.0004964303  0.0021145866          0  0.0021431086  0.002026901  0.0000000000
## 3  0.0070207527  0.0032267283          0  0.0000000000  0.003055398  0.0043286399

```

```

## 18 0.000000e+00 0.0138890153 0.018523137 0.000000e+00 0.0116967639 9.903829e-05
## 19 0.000000e+00 0.0071017233 0.010933577 0.000000e+00 0.00000000000 0.000000e+00
## 20 1.732077e-03 0.0031654063 0.006769595 0.000000e+00 0.00000000000 9.670842e-04
## 21 1.397591e-03 0.00000000000 0.00000000000 0.000000e+00 0.00000000000 0.000000e+00
##       601162.SH' 603501.SH' 601939.SH' 601933.SH' 600837.SH' 601155.SH'
## 1  0.011556860 0.001960081 0.000000000 0.00000000000 0.00000000000 0.000000000
## 2  0.001819375 0.004902849 0.001307206 0.00000000000 0.0030153441 0.01932329
## 3  0.0000000000 0.005517099 0.007491835 0.0033498286 0.00000000000 0.000000000
## 4  0.003840446 0.006059283 0.000000000 0.0005772171 0.0033118627 0.000000000
## 5  0.003798986 0.020584129 0.000000000 0.0029266768 0.00000000000 0.000000000
## 6  0.0000000000 0.007890988 0.005964553 0.00000000000 0.0036585751 0.03346632
## 7  0.0000000000 0.011365820 0.010463148 0.00000000000 0.00000000000 0.01815224
## 8  0.0000000000 0.017408256 0.000000000 0.00000000000 0.0026428004 0.000000000
## 9  0.0000000000 0.027218261 0.000000000 0.0042338144 0.00000000000 0.02238584
## 10 0.0000000000 0.008033225 0.001684633 0.0017784944 0.0028068888 0.03276572
## 11 0.0000000000 0.006880997 0.000000000 0.00000000000 0.00000000000 0.000000000
## 12 0.0000000000 0.010384096 0.006124200 0.00000000000 0.0011030171 0.000000000
## 13 0.0000000000 0.0000000000 0.009015672 0.0051571723 0.00000000000 0.02203211
## 14 0.0000000000 0.0000000000 0.0000000000 0.00000000000 0.00000000000 0.03283700
## 15 0.0000000000 0.0000000000 0.001755539 0.00000000000 0.0009380714 0.03618714
## 16 0.0000000000 0.0000000000 0.005209108 0.0020761783 0.00000000000 0.01070871
## 17 0.0000000000 0.0000000000 0.0000000000 0.0038608268 0.0006630791 0.000000000
## 18 0.0000000000 0.0000000000 0.003423473 0.00000000000 0.00000000000 0.02395856
## 19 0.0000000000 0.006580517 0.005914673 0.00000000000 0.0008009373 0.01691992
## 20 0.0000000000 0.0000000000 0.002742228 0.0052282560 0.00000000000 0.01699720
## 21 0.0000000000 0.0000000000 0.0000000000 0.00000000000 0.00000000000 0.000000000
##       600745.SH' 603195.SH'...301
## 1  0.005910079 0.0130694529
## 2  0.010846364 0.0116282152
## 3  0.005250309 0.0306700352
## 4  0.004167185 0.0167893708
## 5  0.006410444 0.0399975033
## 6  0.005051477 0.0351300734
## 7  0.0000000000 0.0280609901
## 8  0.0000000000 0.0268018073
## 9  0.017957221 0.0233460241
## 10 0.012097107 0.0007607557
## 11 0.003222506 0.0029029531
## 12 0.0000000000 0.00000000000
## 13 0.0000000000 0.00000000000
## 14 0.0000000000 0.00000000000
## 15 0.0000000000 0.00000000000
## 16 0.0000000000 0.00000000000
## 17 0.0000000000 0.00000000000
## 18 0.0000000000 0.0012385940
## 19 0.006444351 0.00000000000
## 20 0.009698206 0.00000000000
## 21 0.004838503 0.0088845207

```

策略 2：平等投资每一只正预测收益率的股票

```

# 创建一个空的矩阵来存储策略数据，也即策略矩阵
strategy_2 <- matrix(0, nrow = 21, ncol = 300)

```

```

# 遍历每一天的数据行
for (i in 1:21) {
  # 获取当前行的预测收益率
  daily_predictions <- predictions[i, ]

  # 找出正收益率
  positive_returns <- daily_predictions[daily_predictions > 0]

  # 计算每只股票的投资比例
  if (sum_positive_returns > 0) {
    strategy_2[i, daily_predictions > 0] <- 1 / length(positive_returns)
  }
}

# 将 strategy 矩阵转换为数据框
strategy_2 <- as.data.frame(strategy_2)

# 设置列名与 predictions 相同
colnames(strategy_2) <- colnames(predictions)

# 避免缺失值影响
strategy_2[is.na(strategy_2)] <- 0

print(strategy_2)

##      000977.SZ' 000768.SZ' 002466.SZ' 000338.SZ' 000157.SZ' 002415.SZ'
## 1  0.006289308 0.000000000 0.006289308          0 0.000000000 0.000000000
## 2  0.005050505 0.005050505 0.005050505          0 0.000000000 0.005050505
## 3  0.006493506 0.000000000 0.006493506          0 0.006493506 0.006493506
## 4  0.004651163 0.004651163 0.004651163          0 0.004651163 0.004651163
## 5  0.006024096 0.006024096 0.006024096          0 0.000000000 0.000000000
## 6  0.005464481 0.005464481 0.005464481          0 0.000000000 0.005464481
## 7  0.005586592 0.000000000 0.005586592          0 0.005586592 0.005586592
## 8  0.005181347 0.005181347 0.005181347          0 0.005181347 0.005181347
## 9  0.007352941 0.007352941 0.000000000          0 0.000000000 0.007352941
## 10 0.005235602 0.005235602 0.005235602          0 0.000000000 0.005235602
## 11 0.005235602 0.005235602 0.005235602          0 0.005235602 0.005235602
## 12 0.005988024 0.005988024 0.005988024          0 0.005988024 0.005988024
## 13 0.005555556 0.000000000 0.000000000          0 0.000000000 0.005555556
## 14 0.008064516 0.008064516 0.008064516          0 0.000000000 0.008064516
## 15 0.006493506 0.006493506 0.000000000          0 0.006493506 0.006493506
## 16 0.005524862 0.005524862 0.005524862          0 0.005524862 0.005524862
## 17 0.007246377 0.000000000 0.000000000          0 0.000000000 0.007246377
## 18 0.006024096 0.006024096 0.006024096          0 0.000000000 0.006024096
## 19 0.005208333 0.005208333 0.005208333          0 0.005208333 0.005208333
## 20 0.004854369 0.004854369 0.004854369          0 0.004854369 0.004854369
## 21 0.006756757 0.000000000 0.006756757          0 0.000000000 0.006756757
##      600085.SH' 002049.SZ' 600011.SH' 002841.SZ' 002736.SZ' 300782.SZ'
## 1  0.006289308 0.006289308 0.000000000 0.000000000 0.006289308 0.006289308
## 2  0.000000000 0.005050505 0.000000000 0.000000000 0.005050505 0.005050505
## 3  0.000000000 0.006493506 0.006493506 0.000000000 0.006493506 0.006493506
## 4  0.004651163 0.004651163 0.004651163 0.000000000 0.004651163 0.004651163
## 5  0.006024096 0.006024096 0.000000000 0.006024096 0.006024096 0.006024096

```

```

## 20 0.004854369 0.004854369 0.004854369 0.004854369 0.004854369 0.004854369
## 21 0.000000000 0.006756757 0.006756757 0.000000000 0.006756757 0.000000000
##   601868.SH' 601766.SH' 601800.SH' 601618.SH' 601162.SH' 603501.SH'
## 1 0.000000000 0.006289308 0.006289308 0.000000000 0.006289308 0.006289308
## 2 0.000000000 0.005050505 0.005050505 0.005050505 0.005050505 0.005050505
## 3 0.000000000 0.000000000 0.006493506 0.006493506 0.000000000 0.006493506
## 4 0.004651163 0.004651163 0.004651163 0.004651163 0.004651163 0.004651163
## 5 0.000000000 0.006024096 0.006024096 0.000000000 0.006024096 0.006024096
## 6 0.000000000 0.000000000 0.005464481 0.000000000 0.000000000 0.005464481
## 7 0.000000000 0.000000000 0.000000000 0.005586592 0.000000000 0.005586592
## 8 0.000000000 0.005181347 0.000000000 0.000000000 0.000000000 0.005181347
## 9 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.007352941
## 10 0.005235602 0.000000000 0.005235602 0.005235602 0.000000000 0.005235602
## 11 0.005235602 0.000000000 0.005235602 0.000000000 0.000000000 0.005235602
## 12 0.005988024 0.000000000 0.005988024 0.000000000 0.000000000 0.005988024
## 13 0.005555556 0.005555556 0.005555556 0.005555556 0.000000000 0.000000000
## 14 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## 15 0.006493506 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## 16 0.005524862 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## 17 0.007246377 0.000000000 0.007246377 0.000000000 0.000000000 0.000000000
## 18 0.006024096 0.000000000 0.006024096 0.006024096 0.000000000 0.000000000
## 19 0.005208333 0.000000000 0.000000000 0.000000000 0.000000000 0.005208333
## 20 0.004854369 0.000000000 0.000000000 0.004854369 0.000000000 0.000000000
## 21 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   601939.SH' 601933.SH' 600837.SH' 601155.SH' 600745.SH' 603195.SH'...301
## 1 0.000000000 0.000000000 0.000000000 0.000000000 0.006289308 0.006289308
## 2 0.005050505 0.000000000 0.005050505 0.005050505 0.005050505 0.005050505
## 3 0.006493506 0.006493506 0.000000000 0.000000000 0.006493506 0.006493506
## 4 0.000000000 0.004651163 0.004651163 0.000000000 0.004651163 0.004651163
## 5 0.000000000 0.006024096 0.000000000 0.000000000 0.006024096 0.006024096
## 6 0.005464481 0.000000000 0.005464481 0.005464481 0.005464481 0.005464481
## 7 0.005586592 0.000000000 0.000000000 0.005586592 0.000000000 0.005586592
## 8 0.000000000 0.000000000 0.005181347 0.000000000 0.000000000 0.005181347
## 9 0.000000000 0.007352941 0.000000000 0.007352941 0.007352941 0.007352941
## 10 0.005235602 0.005235602 0.005235602 0.005235602 0.005235602 0.005235602
## 11 0.000000000 0.000000000 0.000000000 0.000000000 0.005235602 0.005235602
## 12 0.005988024 0.000000000 0.005988024 0.000000000 0.000000000 0.000000000
## 13 0.005555556 0.005555556 0.000000000 0.005555556 0.000000000 0.000000000
## 14 0.000000000 0.000000000 0.000000000 0.008064516 0.000000000 0.000000000
## 15 0.006493506 0.000000000 0.006493506 0.006493506 0.000000000 0.000000000
## 16 0.005524862 0.005524862 0.000000000 0.005524862 0.000000000 0.000000000
## 17 0.000000000 0.007246377 0.007246377 0.000000000 0.000000000 0.000000000
## 18 0.006024096 0.000000000 0.000000000 0.006024096 0.000000000 0.006024096
## 19 0.005208333 0.000000000 0.005208333 0.005208333 0.005208333 0.000000000
## 20 0.004854369 0.004854369 0.000000000 0.004854369 0.004854369 0.000000000
## 21 0.000000000 0.000000000 0.000000000 0.000000000 0.006756757 0.006756757

```

策略 3：平等投资所有股票

```

# 创建一个空的矩阵来存储策略数据，也即策略矩阵
strategy_3 <- matrix(1/300, nrow = 21, ncol = 300)

# 将 strategy 矩阵转换为数据框
strategy_3 <- as.data.frame(strategy_3)

```

```

# 设置列名与 predictions 相同
colnames(strategy_3) <- colnames(predictions)

# 避免缺失值影响
strategy_3[is.na(strategy_3)] <- 0

print(strategy_3)

##      000977.SZ' 000768.SZ' 002466.SZ' 000338.SZ' 000157.SZ' 002415.SZ'
## 1  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 2  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 3  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 4  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 5  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 6  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 7  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 8  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 9  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 10 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 11 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 12 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 13 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 14 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 15 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 16 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 17 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 18 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 19 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 20 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 21 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
##      600085.SH' 002049.SZ' 600011.SH' 002841.SZ' 002736.SZ' 300782.SZ'
## 1  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 2  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 3  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 4  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 5  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 6  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 7  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 8  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 9  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 10 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 11 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 12 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 13 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 14 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 15 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 16 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 17 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 18 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 19 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 20 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 21 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
##      002304.SZ' 002812.SZ' 300628.SZ' 002252.SZ' 002001.SZ' 600019.SH'

```

```

## 15 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 16 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 17 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 18 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 19 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 20 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 21 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
##       601939.SH' 601933.SH' 600837.SH' 601155.SH' 600745.SH' 603195.SH'...301
## 1  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 2  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 3  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 4  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 5  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 6  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 7  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 8  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 9  0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 10 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 11 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 12 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 13 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 14 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 15 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 16 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 17 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 18 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 19 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 20 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333
## 21 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333 0.003333333

```

基于上面的投资策略矩阵，绘制不同策略的资产变化图。

```

# 避免缺失值影响
actual_returns[is.na(actual_returns)] <- 0

# 创建三个向量来存储 21 天的资产值变动
initial_value <- 10000

asset_values_1 <- numeric(22)
asset_values_1[1] <- initial_value

asset_values_2 <- numeric(22)
asset_values_2[1] <- initial_value

asset_values_3 <- numeric(22)
asset_values_3[1] <- initial_value

# 计算每天的资产值
for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_1[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_1[i+1] <- asset_values_1[i] * (1 + daily_return_rate)
  asset_values_2[i+1] <- asset_values_2[i] * (1 + daily_return_rate)
  asset_values_3[i+1] <- asset_values_3[i] * (1 + daily_return_rate)
}

```

```

    asset_values_1[i + 1] <- asset_values_1[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_2[,]
  daily_returns <- actual_returns[,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_2[i + 1] <- asset_values_2[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_3[,]
  daily_returns <- actual_returns[,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_3[i + 1] <- asset_values_3[i] * (1 + daily_return_rate)
}

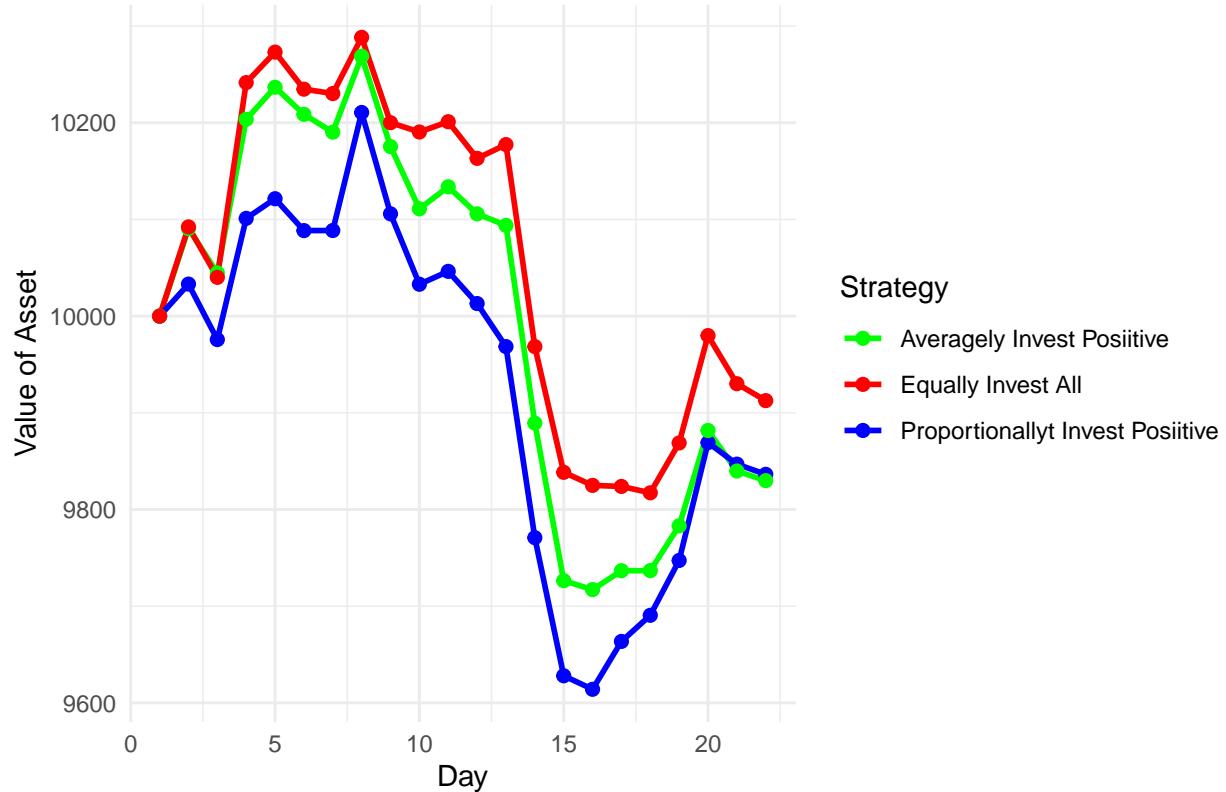
# 创建数据框用于绘图
plot_asset_values <- data.frame(
  Day = rep(1:22, 3),
  AssetValue = c(asset_values_1, asset_values_2, asset_values_3),
  Strategy = rep(c("Proportionallyt Invest Posiitive", "Averagely Invest Posiitive", "Equally Invest All"),
)

# 绘制图形
library(ggplot2)

ggplot(plot_asset_values, aes(x = Day, y = AssetValue, color = Strategy, group = Strategy)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  ggtitle("21 Day Changes in the Value of Asset") +
  xlab("Day") +
  ylab("Value of Asset") +
  theme_minimal() +
  scale_color_manual(values = c("Proportionallyt Invest Posiitive" = "blue", "Averagely Invest Posiitive" = "green", "Equally Invest All" = "red"))
}

```

21 Day Changes in the Value of Asset



我们作如下总结：值得注意的是，按照 arma 模型预测的指导进行的股票投资并没有得到期待的更好表现。相反，平均投资所有资产的策略甚至表现要优于我们的时间序列预测指导的两种策略进行的投资。这也说明了 arma 模型对于预测股票收益率能力十分有限，难以准确预测，进而以 arma 模型指导的股票投资策略也表现的不如人意。

这也回答了 1. Why not invest all your money, say 10000 RMB in stock B? 也即：首先 arma 模型对于预测股票收益率能力十分有限；其次即使 arma 模型可以较好预测，投资单独一个股票即使期望收益高，但是风险极大。综合看来，投资所有股票到预测的最高收益的股票是不恰当的，这会带来极大的风险。

接下来我们探索尝试一些自己的策略。策略 4 是按照马科维茨最优组合理论进行的投资，我们的股票期望值直接选取训练集历史期望值，并且根据训练集得到股票的协方差矩阵。这种投资每天不改变投资比例，即在月初固定投资策略比例。策略 5 也是按照马科维茨理论的指导，区别在于我们股票期望分别采用每天的预测值，这样每天更新每只股票的投资的比例。结果如下所示：

策略 4：马科维茨最优组合

```
history_data <- data %>% filter(date <= as.Date("2022-11-30"))
# 将日期列排除，并确保所有列都是数值型
# 假设 history_data 是你的数据框，第一列是日期列
# 将日期列排除，并确保所有列都是数值型
numeric_data <- history_data %>%
  select(-1) %>%
  mutate(across(everything(), as.numeric))

# 去除缺失值
numeric_data <- na.omit(numeric_data)

# 计算期望收益率和协方差矩阵
```

```

expected_returns <- colMeans(numeric_data)
cov_matrix <- cov(numeric_data)

# 添加小的常数到协方差矩阵的对角线
make_positive_definite <- function(cov_matrix, epsilon = 1e-8) {
  diag(cov_matrix) <- diag(cov_matrix) + epsilon
  return(cov_matrix)
}
cov_matrix <- make_positive_definite(cov_matrix)

# 求最优解
optimize_sharpe_ratio <- function(expected_return, cov_return, risk_free_rate) {
  # 定义 Sharpe 比率的目标函数
  sharpe_ratio <- function(weights) {
    portfolio_return <- sum(weights * expected_return)
    portfolio_volatility <- sqrt(as.numeric(t(weights) %*% cov_return %*% weights))
    -((portfolio_return - risk_free_rate) / portfolio_volatility) # 取负值以进行最大化
  }

  # 初始权重（假设等权重）
  initial_weights <- rep(1 / length(expected_return), length(expected_return))

  # 适应度函数
  fitness <- function(weights) {
    # 保证权重和为 1
    weights <- weights / sum(weights)
    sharpe_ratio(weights)
  }

  # 使用遗传算法进行优化
  ga_result <- ga(
    type = "real-valued",
    fitness = fitness,
    min = rep(0, length(expected_return)),
    max = rep(1, length(expected_return)),
    popSize = 50,
    maxiter = 1000,
    run = 100,
    pmutation = 0.2
  )

  # 获取最优权重
  optimal_weights <- ga_result@solution
  optimal_weights <- optimal_weights / sum(optimal_weights) # 归一化

  return(optimal_weights)
}
# 最优化函数

```

```

risk_free_rate <- 0 # 无风险利率，可以根据市场情况调整

# 使用函数求解最优权重向量
optimal_weights <- optimize_sharpe_ratio(expected_returns, cov_matrix, risk_free_rate)

# 创建一个零矩阵
strategy_4 <- matrix(0, nrow = 21, ncol = 300)

# 确保 optimized_weights 的长度是否与 strategy_4 的列数一致
if (length(optimal_weights) == ncol(strategy_4)) {
  # 将每一行都填充为 optimized_weights
  for (i in 1:nrow(strategy_4)) {
    strategy_4[i, ] <- optimal_weights
  }
} else {
  stop("optimized_weights 的长度与 strategy_4 的列数不一致。")
}

# 将 strategy 矩阵转换为数据框
strategy_4 <- as.data.frame(strategy_4)

# 设置列名与 predictions 相同
colnames(strategy_4) <- colnames(predictions)

# 避免缺失值影响
strategy_4[is.na(strategy_4)] <- 0

print(strategy_4)

##      000977.SZ'  000768.SZ'  002466.SZ'  000338.SZ'  000157.SZ'  002415.SZ'
## 1  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 2  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 3  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 4  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 5  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 6  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 7  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 8  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 9  0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 10 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 11 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 12 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 13 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 14 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 15 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 16 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 17 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 18 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 19 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 20 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
## 21 0.003315378 0.001421311 0.001306702 0.004498423 0.002374564 0.005490263
##       600085.SH'  002049.SZ'  600011.SH'  002841.SZ'  002736.SZ'  300782.SZ'

```

```

## 3      0.004833346
## 4      0.004833346
## 5      0.004833346
## 6      0.004833346
## 7      0.004833346
## 8      0.004833346
## 9      0.004833346
## 10     0.004833346
## 11     0.004833346
## 12     0.004833346
## 13     0.004833346
## 14     0.004833346
## 15     0.004833346
## 16     0.004833346
## 17     0.004833346
## 18     0.004833346
## 19     0.004833346
## 20     0.004833346
## 21     0.004833346

# 避免缺失值影响
actual_returns[is.na(actual_returns)] <- 0

# 创建三个向量来存储 21 天的资产值变动
initial_value <- 10000

asset_values_1 <- numeric(22)
asset_values_1[1] <- initial_value

asset_values_2 <- numeric(22)
asset_values_2[1] <- initial_value

asset_values_3 <- numeric(22)
asset_values_3[1] <- initial_value

asset_values_4 <- numeric(22)
asset_values_4[1] <- initial_value

# 计算每天的资产值
for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_1[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_1[i + 1] <- asset_values_1[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_2[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
}

```

```

    asset_values_2[i + 1] <- asset_values_2[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_3[,]
  daily_returns <- actual_returns[,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_3[i + 1] <- asset_values_3[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_4[,]
  daily_returns <- actual_returns[,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_4[i + 1] <- asset_values_4[i] * (1 + daily_return_rate)
}

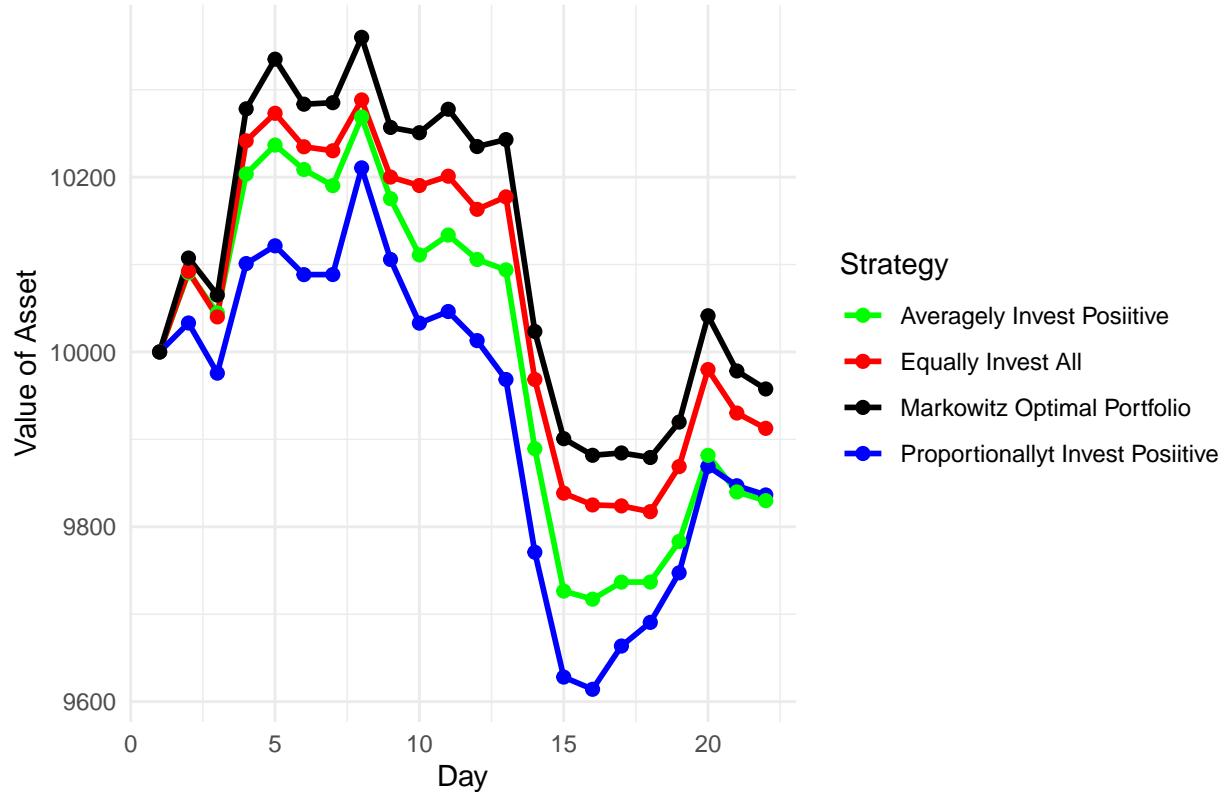
# 创建数据框用于绘图
plot_asset_values <- data.frame(
  Day = rep(1:22, 4),
  AssetValue = c(asset_values_1, asset_values_2, asset_values_3, asset_values_4),
  Strategy = rep(c("Proportionallyt Invest Posiitive", "Averagely Invest Posiitive", "Equally Invest All Assets", "Randomly Invest Assets"), 4)
)

# 绘制图形
library(ggplot2)

ggplot(plot_asset_values, aes(x = Day, y = AssetValue, color = Strategy, group = Strategy)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  ggtitle("21 Day Changes in the Value of Asset") +
  xlab("Day") +
  ylab("Value of Asset") +
  theme_minimal() +
  scale_color_manual(values = c("Proportionallyt Invest Posiitive" = "blue", "Averagely Invest Posiitive" = "orange", "Equally Invest All Assets" = "green", "Randomly Invest Assets" = "red"))

```

21 Day Changes in the Value of Asset



策略 5：马科维茨最优组合 +ARMA 预测

```
history_data <- data %>% filter(date <= as.Date("2022-11-30"))
# 将日期列排除，并确保所有列都是数值型
# 假设 history_data 是你的数据框，第一列是日期列
# 将日期列排除，并确保所有列都是数值型
numeric_data <- history_data %>%
  select(-1) %>%
  mutate(across(everything(), as.numeric))

# 去除缺失值
numeric_data <- na.omit(numeric_data)

# 计算期望收益率和协方差矩阵
cov_matrix <- cov(numeric_data)

# 添加小的常数到协方差矩阵的对角线
make_positive_definite <- function(cov_matrix, epsilon = 1e-8) {
  diag(cov_matrix) <- diag(cov_matrix) + epsilon
  return(cov_matrix)
}
cov_matrix <- make_positive_definite(cov_matrix)
```

```

# 求最优解
optimize_sharpe_ratio <- function(expected_return, cov_return, risk_free_rate) {
  # 定义 Sharpe 比率的目标函数
  sharpe_ratio <- function(weights) {
    portfolio_return <- sum(weights * expected_return)
    portfolio_volatility <- sqrt(as.numeric(t(weights) %*% cov_return %*% weights))
    -((portfolio_return - risk_free_rate) / portfolio_volatility) # 取负值以进行最大化
  }

  # 初始权重（假设等权重）
  initial_weights <- rep(1 / length(expected_return), length(expected_return))

  # 适应度函数
  fitness <- function(weights) {
    # 保证权重和为 1
    weights <- weights / sum(weights)
    sharpe_ratio(weights)
  }

  # 使用遗传算法进行优化
  ga_result <- ga(
    type = "real-valued",
    fitness = fitness,
    min = rep(0, length(expected_return)),
    max = rep(1, length(expected_return)),
    popSize = 50,
    maxiter = 1000,
    run = 100,
    pmutation = 0.2
  )

  # 获取最优权重
  optimal_weights <- ga_result@solution
  optimal_weights <- optimal_weights / sum(optimal_weights) # 归一化

  return(optimal_weights)
}

# 最优化函数

risk_free_rate <- 0 # 无风险利率，可以根据市场情况调整

strategy_5 <- matrix(0, nrow = 21, ncol = 300)

for (i in 1:21) {
  expected_returns <- predictions[i, ]
  print(paste("Day", i, "Expected Returns:", paste(expected_returns, collapse = ", "))) # 调试信息
  optimal_weights <- optimize_sharpe_ratio(expected_returns, cov_matrix, risk_free_rate)
  strategy_5[i, ] <- optimal_weights
}

```

```

## [1] "Day 1 Expected Returns: 0.128551945187706, -0.0391232467713953, 0.429058467040995, 0, -0.193497
## [1] "Day 2 Expected Returns: 0.287401007814809, 0.110573865827987, 0.105552412150689, 0, -0.104264663
## [1] "Day 3 Expected Returns: 0.134813675191109, -0.225010126159585, 0.0296194360940273, 0, 0.18087788
## [1] "Day 4 Expected Returns: 0.196902443159214, 0.31726392532179, 0.171268392627828, 0, 0.1154609681
## [1] "Day 5 Expected Returns: 0.104216961320784, 0.0813989719892281, 0.276916051794393, 0, -0.23930347
## [1] "Day 6 Expected Returns: 0.104007397483444, 0.0204203400239029, 0.212396221902123, 0, -0.10874620
## [1] "Day 7 Expected Returns: 0.0739657109789958, -0.23232607110533, 0.0926485394751263, 0, 0.23433793
## [1] "Day 8 Expected Returns: 0.108137027583973, 0.154731844613893, 0.168375536907232, 0, 0.107266861
## [1] "Day 9 Expected Returns: 0.181152954443199, 0.302440814221579, -0.060893589621944, 0, -0.24308904
## [1] "Day 10 Expected Returns: 0.0289811680079377, 0.121886551554839, 0.057797639872331, 0, -0.0423288
## [1] "Day 11 Expected Returns: 0.14797774904639, 0.120545988784091, 0.169528837226855, 0, 0.213071846
## [1] "Day 12 Expected Returns: 0.157429377054416, 0.240630593257649, 0.358922526014456, 0, 0.059498377
## [1] "Day 13 Expected Returns: 0.064584120336705, -0.038952406387909, -0.277413077822936, 0, -0.15110
## [1] "Day 14 Expected Returns: 0.0541978524386428, 0.00608835228901508, 0.678607515340774, 0, -0.08141
## [1] "Day 15 Expected Returns: 0.0635113753481309, 0.127400088809787, -0.262532797902067, 0, 0.2102430
## [1] "Day 16 Expected Returns: 0.092896054880414, 0.130094307702784, 0.0365732571833363, 0, 0.1137905
## [1] "Day 17 Expected Returns: 0.115808230961881, -0.141791504227168, -0.104944828639696, 0, -0.17748
## [1] "Day 18 Expected Returns: 0.185389979583047, 0.0968344526292159, 0.207901700591938, 0, -0.079593
## [1] "Day 19 Expected Returns: 0.181883765115921, 0.51752070998543, 0.358245577005997, 0, 0.165484986
## [1] "Day 20 Expected Returns: 0.125886745331018, 0.269955932739423, 0.163603477284151, 0, 0.05730410
## [1] "Day 21 Expected Returns: 0.0388664948067443, -0.13108485566875, 0.0149084307471525, 0, -0.16094

# 将 strategy 矩阵转换为数据框
strategy_5 <- as.data.frame(strategy_5)

# 设置列名与 predictions 相同
colnames(strategy_5) <- colnames(predictions)

# 避免缺失值影响
strategy_5[is.na(strategy_5)] <- 0

print(strategy_5)

##      000977.SZ' 000768.SZ' 002466.SZ' 000338.SZ' 000157.SZ' 002415.SZ'
## 1  0.001943715 0.003715004 0.0010407697 0.002283112 0.0054014655 0.002494196
## 2  0.001479599 0.002651171 0.0045334203 0.003946008 0.0064779386 0.003384117
## 3  0.002160100 0.005506352 0.0015969181 0.001646430 0.0009232030 0.002121170
## 4  0.002473504 0.002794316 0.0017153168 0.004785245 0.0023426043 0.002358314
## 5  0.003224594 0.001935730 0.0026845416 0.003328051 0.0056481922 0.003517041
## 6  0.003533064 0.002064388 0.0018036429 0.002933545 0.0040214409 0.002157745
## 7  0.003624405 0.007031668 0.0033956400 0.004122694 0.0023511915 0.001796916
## 8  0.002429117 0.003417757 0.0019334093 0.003601417 0.0027491736 0.001608461
## 9  0.001447256 0.001945163 0.0030138446 0.002024055 0.0050270393 0.002467068
## 10 0.001870299 0.002720962 0.0019392904 0.004061826 0.0054037395 0.002045219
## 11 0.002563500 0.003019878 0.0031153183 0.004210958 0.0008865358 0.001714457
## 12 0.002777925 0.001751212 0.0010070261 0.003749372 0.0017703066 0.002760091
## 13 0.001614337 0.004435474 0.0071776226 0.002805451 0.0065813607 0.002550156
## 14 0.001646228 0.003151959 0.0009750663 0.002161511 0.0020934082 0.002843326
## 15 0.001575263 0.002607955 0.0043225589 0.002397147 0.0016730276 0.002144302
## 16 0.002023078 0.001736564 0.0032036524 0.002319867 0.0027091584 0.001334121
## 17 0.003969438 0.005181058 0.0033377030 0.001859467 0.0028309868 0.001884698
## 18 0.001008973 0.003154881 0.0012625277 0.003745277 0.0040927910 0.001183322
## 19 0.002415097 0.001308305 0.0016537909 0.005758049 0.0015010878 0.004228196

```

```

##      600745.SH' 603195.SH'...301
## 1  0.0014300043  0.0001115990
## 2  0.0005264924  0.0009001434
## 3  0.0022046883  0.0003710799
## 4  0.0009786816  0.0016702138
## 5  0.0018149254  0.0006685342
## 6  0.0016346131  0.0003445809
## 7  0.0050585881  0.0003947829
## 8  0.0024950478  0.0009021019
## 9  0.0012239889  0.0014748809
## 10 0.0005534238  0.0037846671
## 11 0.0038751649  0.0031092289
## 12 0.0044414379  0.0070586316
## 13 0.0049522528  0.0077736160
## 14 0.0048282527  0.0076248200
## 15 0.0033229458  0.0073276557
## 16 0.0038357005  0.0077626037
## 17 0.0018760120  0.0065653594
## 18 0.0023864902  0.0015883006
## 19 0.0014863540  0.0041450527
## 20 0.0020128209  0.0063846807
## 21 0.0015598020  0.0015255550

# 避免缺失值影响
actual_returns[is.na(actual_returns)] <- 0

# 创建三个向量来存储 21 天的资产值变动
initial_value <- 10000

asset_values_1 <- numeric(22)
asset_values_1[1] <- initial_value

asset_values_2 <- numeric(22)
asset_values_2[1] <- initial_value

asset_values_3 <- numeric(22)
asset_values_3[1] <- initial_value

asset_values_4 <- numeric(22)
asset_values_4[1] <- initial_value

asset_values_5 <- numeric(22)
asset_values_5[1] <- initial_value

# 计算每天的资产值
for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_1[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_1[i + 1] <- asset_values_1[i] * (1 + daily_return_rate)
}

```

```

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_2[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_2[i + 1] <- asset_values_2[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_3[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_3[i + 1] <- asset_values_3[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_4[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_4[i + 1] <- asset_values_4[i] * (1 + daily_return_rate)
}

for (i in 1:21) {
  # 计算当天的实际收益率
  daily_strategy <- strategy_5[i,]
  daily_returns <- actual_returns[i,]
  daily_return_rate <- sum(daily_strategy * daily_returns) / 100
  # 计算当天的资产值
  asset_values_5[i + 1] <- asset_values_5[i] * (1 + daily_return_rate)
}

# 创建数据框用于绘图
plot_asset_values <- data.frame(
  Day = rep(1:22, 5),
  AssetValue = c(asset_values_1, asset_values_2, asset_values_3, asset_values_4, asset_values_5),
  Strategy = rep(c("Proportionallyt Invest Posiitive", "Averagely Invest Posiitive", "Equally Invest All"),
)

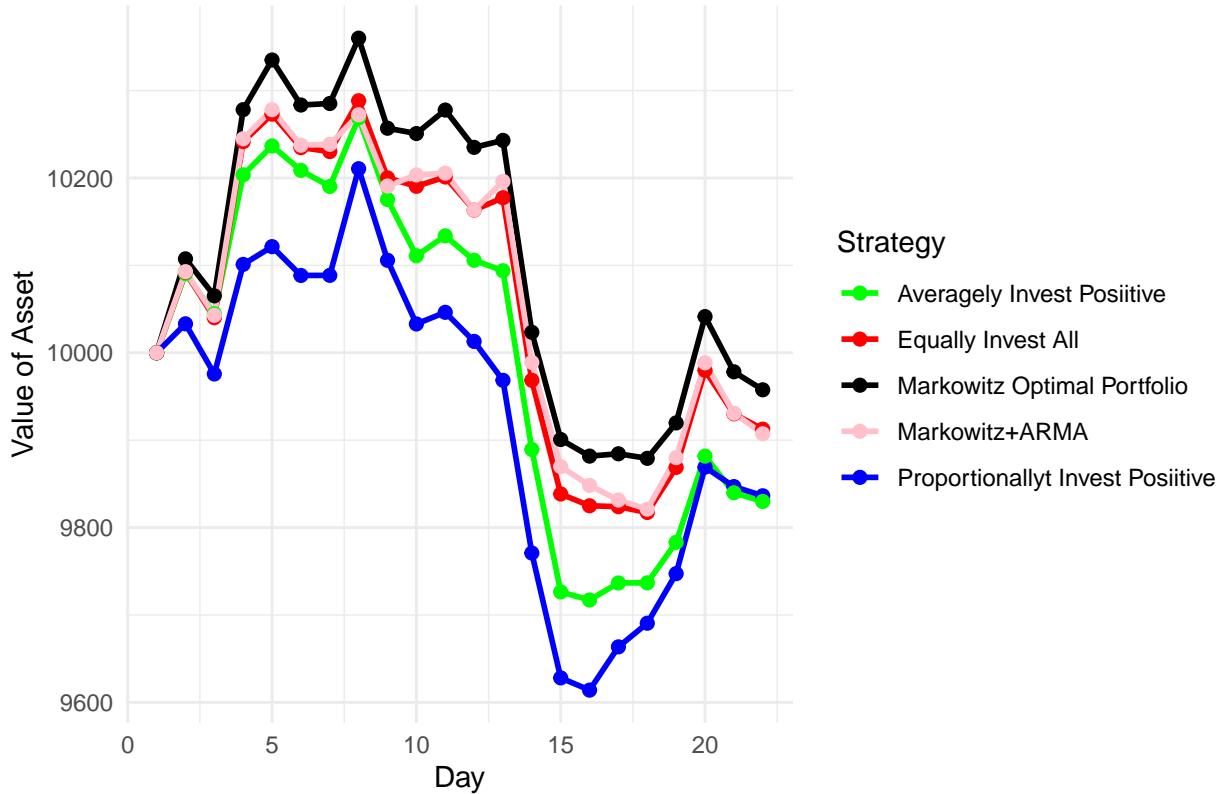
# 绘制图形
library(ggplot2)

ggplot(plot_asset_values, aes(x = Day, y = AssetValue, color = Strategy, group = Strategy)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  ggtitle("21 Day Changes in the Value of Asset") +
  xlab("Day") +
  ylab("Value of Asset")

```

```
theme_minimal() +
scale_color_manual(values = c("Proportionallyt Invest Posiitive" = "blue", "Averagely Invest Posiitive" = "red", "Markowitz Optimal Portfolio" = "black", "Markowitz+ARMA" = "pink", "Equally Invest All" = "green"))
```

21 Day Changes in the Value of Asset



我们做如下总结：可以看到，马科维茨最优组合理论指导下的资产配置相对于其他策略而言表现是更加优秀的，它追求了更高的夏普比率。黑色的线代表一次性固定的马科维茨策略，并没有用到时间序列预测，而可见其趋势与按比例投资等基于预测的策略相似，但是每一天的资产值都更高，即表现更优秀。而粉色的线条是马科维茨理论 +ARMA 预测的策略，可以看出，这一策略的风格与其他策略不同，它的整体波动相比其他策略都更低，也即相对更加稳健。最终月末的时候，反而在所有策略里获得了最高的资产价值，这表明在市场存在下行风险的时候，这一策略似乎更加稳健。

END

第二题：日本收入不平等研究

吕欣远

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import itertools
```

Part A

Pre process

```
In [2]: df=pd.read_csv('C:/Users/14857/Desktop/时序分析/final project(1)/final project/TS project.csv')
df.head()
```

Out[2]:

Time	Year	Quarterly % Change of Inflation (quarterly)	Real GDP Change	Nominal GDP Change	Nikkei 225 Index	Quarterly Nikkei 225 Index Change	Monetary Base	Monetary Base (SA)	Quarterly Change MB	% Quarterly Change- MB (SA)	Income Inequality (disposable income)	Empl
0	1	1994	NaN	1.2	1.0	19474.75787	NaN	417272.66667	413229.0000	NaN	NaN	0.285403
1	2	1994	NaN	-0.6	-0.2	20397.33508	0.047373	413218.3333	417997.66667	-0.009716	0.011540	0.243823
2	3	1994	NaN	1.2	0.8	20369.57046	-0.001361	422350.66667	425975.66667	0.022101	0.019086	0.251858
3	4	1994	NaN	-0.4	-0.3	19473.21672	-0.044005	431003.66667	427084.66667	0.020488	0.002603	0.208498
4	5	1995	-0.62	1.1	0.7	17756.83164	-0.088141	435148.0000	430382.66667	0.009616	0.007722	0.277298

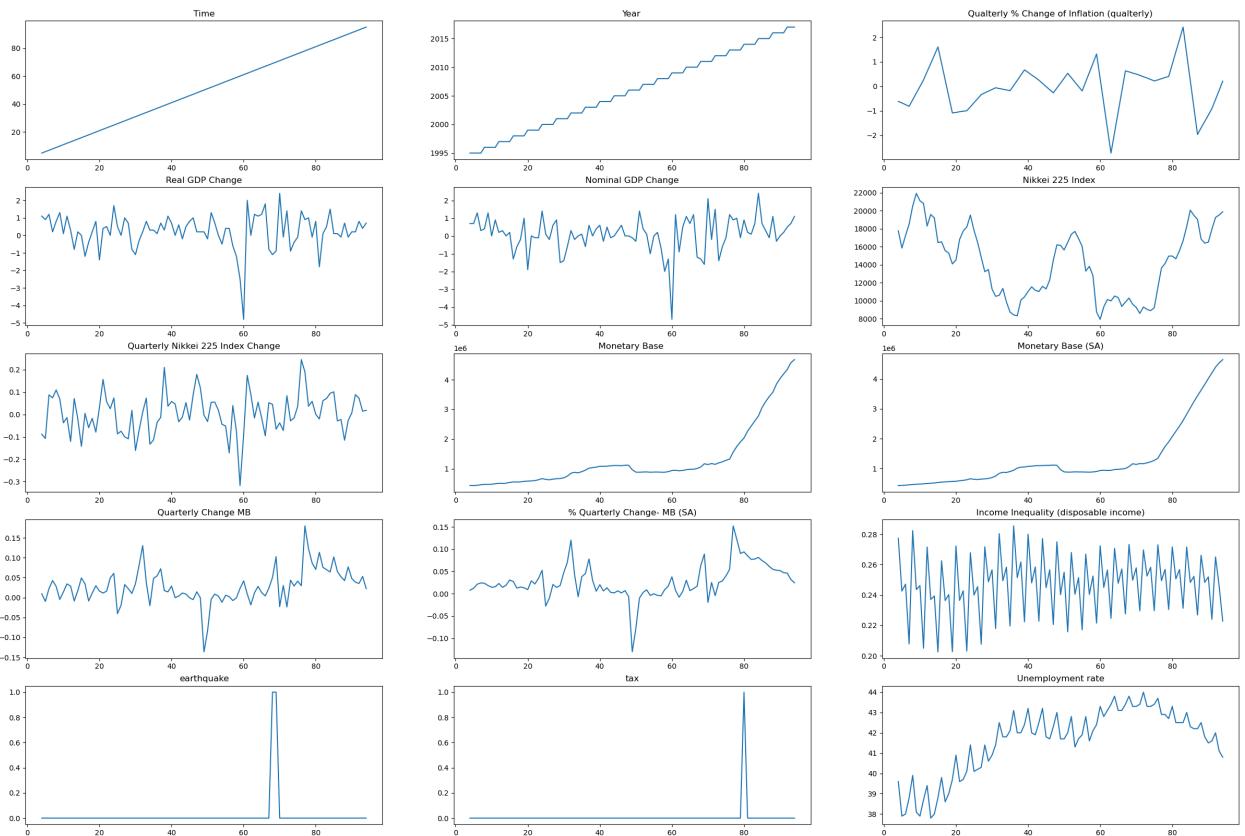
```
In [3]: #Name the columns
inflation_col="Quarterly % Change of Inflation (quarterly)"
gdp_change_col="Real GDP Change"
index_col="Nikkei 225 Index"
index_change_col="Quarterly Nikkei 225 Index Change"
mb_col="Monetary Base (SA)"
mb_change_col="% Quarterly Change- MB (SA)"
income_inequal_col="Income Inequality (disposable income)"
unemployment_rate_col="Unemployment rate"
earthquake_col="earthquake"
tax_col="tax"

#delete the Nah rows
df=df.dropna(axis=0, how='any')
print(len(df))
#Gen the unemployment rate
df['Unemployment rate']=100-df['Employment rate']
```

91

Plot the series.

```
In [4]: cols=list(df.columns)
cols.remove("Employment rate")
fig, axs=plt.subplots(5, 3, figsize=(30, 20))
for i,col in enumerate(cols):
    axs[i//3, i%3].plot(df[col])
    axs[i//3, i%3].set_title(col)
plt.show()
```

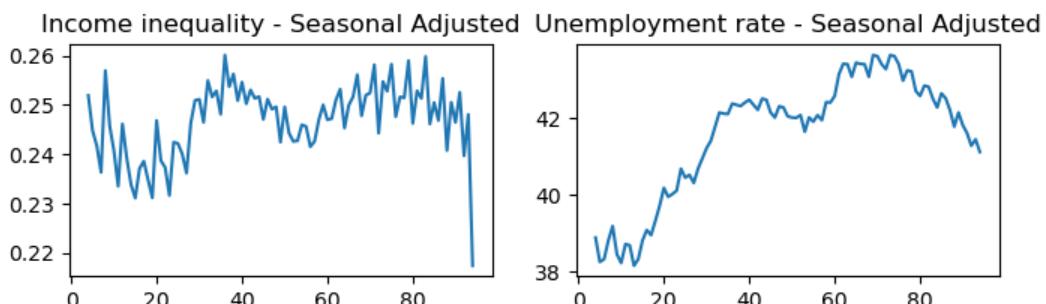


Adjust the series which have seasonality.

```
In [5]: from statsmodels.tsa.seasonal import seasonal_decompose
#decompose the seasonality
sd=seasonal_decompose(df[income_inequal_col], period=4, model="additive")
income_inequal_sa_col="Income inequality - Seasonal Adjusted"
df[income_inequal_sa_col]=df[income_inequal_col]-sd.seasonal

sd=seasonal_decompose(df[unemployment_rate_col], period=4, model="additive")
unemployment_sa_col="Unemployment rate - Seasonal Adjusted"
df[unemployment_sa_col]=df[unemployment_rate_col]-sd.seasonal

#plot the adjusted series
fig, axs=plt.subplots(1, 2, figsize=(8, 2))
axs[0].plot(df[income_inequal_sa_col])
axs[0].set_title(income_inequal_sa_col)
axs[1].plot(df[unemployment_sa_col])
axs[1].set_title(unemployment_sa_col)
plt.show()
```



Test the time trend and do the ADF test.

The Series which will be used in later:

```
inflation_col; gdp_change_col; index_col; index_change_col; mb_col; mb_change_col;  
income_inequal_sa_col; unemployment_sa_col; earthquake_col; tax_col;
```

```
In [6]: from statsmodels.tsa.stattools import adfuller  
from scipy.stats import linregress  
  
target_cols=[inflation_col, gdp_change_col, index_col, index_change_col, mb_col, mb_change_col, income_inequal_sa_col, \  
            unemployment_sa_col, earthquake_col, tax_col]  
len(target_cols)  
  
adf_tab=pd.DataFrame(columns=['Name', 'tau', 'p-value', '5%', 'Lags', 'Obs', 'Trend'])
```

```
In [7]: from scipy.stats import linregress
from statsmodels.tsa.stattools import adfuller

def adf_test(data, adf_tab, name):
    _, _, p_value, _ = linregress(df["Time"], data)
    mode = "ct" if p_value < 0.05 else "c"
    print("-----")
    print(mode)
    result = adfuller(data, regression=mode)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print(' %ts: %.3f' % (key, value))
    trend = 'No' if mode == 'c' else 'Yes'
    stationary = 'No' if result[1] > 0.05 else 'Yes'

    # Create a new DataFrame for this row and concatenate it with adf_tab
    new_row = pd.DataFrame({
        'Name': [name],
        'tau': [round(result[0], 2)],
        'p-Value': [round(result[1], 2)],
        '5%': [round(result[4]['5%'], 2)],
        'Lags': [result[2]],
        'Obs': [result[3]],
        'Trend': [trend],
        'Stationary': [stationary]
    })
    adf_tab = pd.concat([adf_tab, new_row], ignore_index=True)
    return adf_tab

# Assuming df and target_cols are defined earlier
adf_tab = pd.DataFrame() # Initialize DataFrame to store results
for col in target_cols:
    adf_tab = adf_test(df[col], adf_tab, col)

display(adf_tab.set_index('Name'))
```

c
ADF Statistic: -3.530350
p-value: 0.007241
Critical Values:
1%: -3.514
5%: -2.898
10%: -2.586

c
ADF Statistic: -5.807464
p-value: 0.000000
Critical Values:
1%: -3.507
5%: -2.895
10%: -2.585

c
ADF Statistic: -1.843177
p-value: 0.359246
Critical Values:
1%: -3.506
5%: -2.895
10%: -2.584

c
ADF Statistic: -6.690223
p-value: 0.000000
Critical Values:
1%: -3.505
5%: -2.894
10%: -2.584

ct
ADF Statistic: -0.006960
p-value: 0.994162
Critical Values:
1%: -4.064
5%: -3.461
10%: -3.157

ct
ADF Statistic: -4.321440
p-value: 0.002913
Critical Values:
1%: -4.063
5%: -3.460
10%: -3.156

ct
ADF Statistic: -1.478231
p-value: 0.836467
Critical Values:
1%: -4.077
5%: -3.467
10%: -3.160

ct
ADF Statistic: -0.283630
p-value: 0.989885
Critical Values:
1%: -4.071
5%: -3.464
10%: -3.158

c
ADF Statistic: -4.692389
p-value: 0.000087
Critical Values:
1%: -3.508
5%: -2.895
10%: -2.585

c
ADF Statistic: -9.486833
p-value: 0.000000
Critical Values:
1%: -3.505
5%: -2.894
10%: -2.584

Name	tau	p-Value	5%	Lags	Obs	Trend	Stationary
Quarterly % Change of Inflation (qualterly)	-3.53	0.01	-2.90	9	81	No	Yes
Real GDP Change	-5.81	0.00	-2.89	2	88	No	Yes
Nikkei 225 Index	-1.84	0.36	-2.89	1	89	No	No
Quarterly Nikkei 225 Index Change	-6.69	0.00	-2.89	0	90	No	Yes
Monetary Base (SA)	-0.01	0.99	-3.46	1	89	Yes	No
% Quarterly Change- MB (SA)	-4.32	0.00	-3.46	0	90	Yes	Yes
Income inequality - Seasonal Adjusted	-1.48	0.84	-3.47	10	80	Yes	No
Unemployment rate - Seasonal Adjusted	-0.28	0.99	-3.46	6	84	Yes	No
earthquake	-4.69	0.00	-2.90	3	87	No	Yes
tax	-9.49	0.00	-2.89	0	90	No	Yes

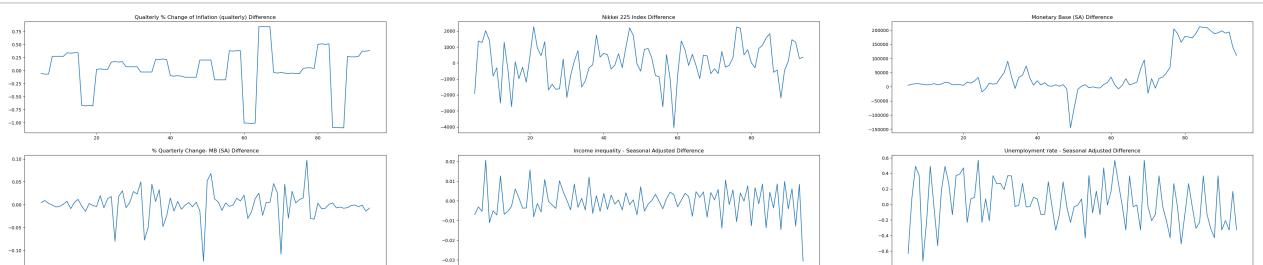
Do the first order difference.

For the non-stationary series and series which are stationary but have time trend, we do first difference and for the interpretability, we just do first difference without apply higher difference.

```
In [8]: inflation_diff_col="Quarterly % Change of Inflation (qualterly) Difference"
index_diff_col="Nikkei 225 Index Difference"
mb_diff_col="Monetary Base (SA) Difference"
mb_change_diff_col="% Quarterly Change- MB (SA) Difference"
income_inequal_diff_col="Income inequality - Seasonal Adjusted Difference"
unemployment_diff_col="Unemployment rate - Seasonal Adjusted Difference"

df[inflation_diff_col]=df[inflation_col].diff()
df[index_diff_col]=df[index_col].diff()
df[mb_diff_col]=df[mb_col].diff()
df[mb_change_diff_col]=df[mb_change_col].diff()
df[income_inequal_diff_col]=df[income_inequal_sa_col].diff()
df[unemployment_diff_col]=df[unemployment_sa_col].diff()

cols=[inflation_diff_col, index_diff_col, mb_diff_col, mb_change_diff_col, income_inequal_diff_col, unemployment_diff_col]
fig, axs=plt.subplots(2, 3, figsize=(50, 10))
for i,col in enumerate(cols):
    axs[i//3, i%3].plot(df[col])
    axs[i//3, i%3].set_title(col)
plt.show()
```



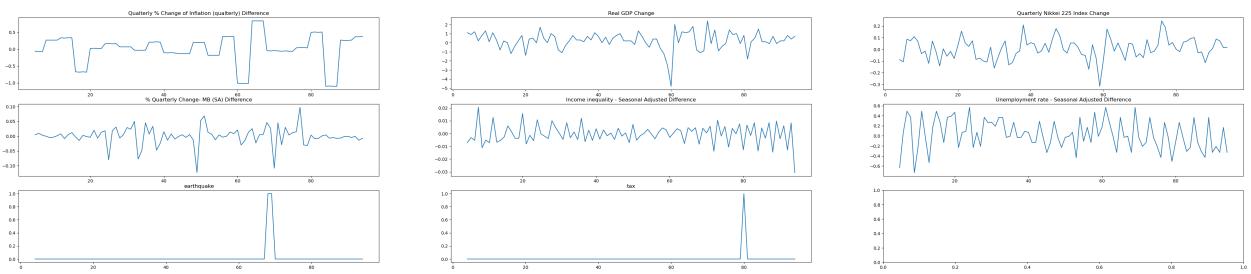
Use AIC/BIC to decide p in VAR(p).

Since `mb_change_diff_col` has the same information as `mb_change_diff_col` and `index_change_col` as `index_diff_col`, we just use `mb_change_diff_col` and `index_change_col`. And We plot the series used in VAR.

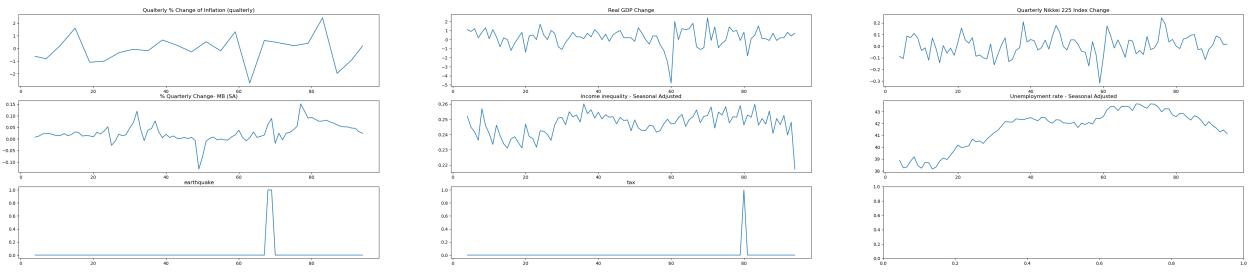
```
In [9]: var_data1 = df[[inflation_diff_col, gdp_change_col, index_change_col, mb_change_diff_col, income_inequal_diff_col, \
                    unemployment_diff_col, earthquake_col, tax_col]]
var_data1 = var_data1.dropna(axis=0, how="any")
var_data2 = df[[inflation_col, gdp_change_col, index_change_col, mb_change_col, income_inequal_sa_col, unemployment_sa_col, \
                    earthquake_col, tax_col]]
var_data2 = var_data2.dropna(axis=0, how="any")
```

VAR_1 use the stationary series which with first order differnece process. And VAR_2 use the series with out differnce for it will be more easy to explain.

```
In [10]: cols=[inflation_diff_col,gdp_change_col,index_change_col,mb_change_diff_col,income_inequal_diff_col,unemployment_diff_col]
fig,axs=plt.subplots(3,3,figsize=(50,10))
for i,col in enumerate(cols):
    axs[i//3,i%3].plot(df[col])
    axs[i//3,i%3].set_title(col)
plt.show()
```



```
In [11]: cols=[inflation_col,gdp_change_col,index_change_col,mb_change_col,income_inequal_sa_col,unemployment_sa_col,earthquake_col]
fig,axs=plt.subplots(3,3,figsize=(50,10))
for i,col in enumerate(cols):
    axs[i//3,i%3].plot(df[col])
    axs[i//3,i%3].set_title(col)
plt.show()
```



The Model selection process.

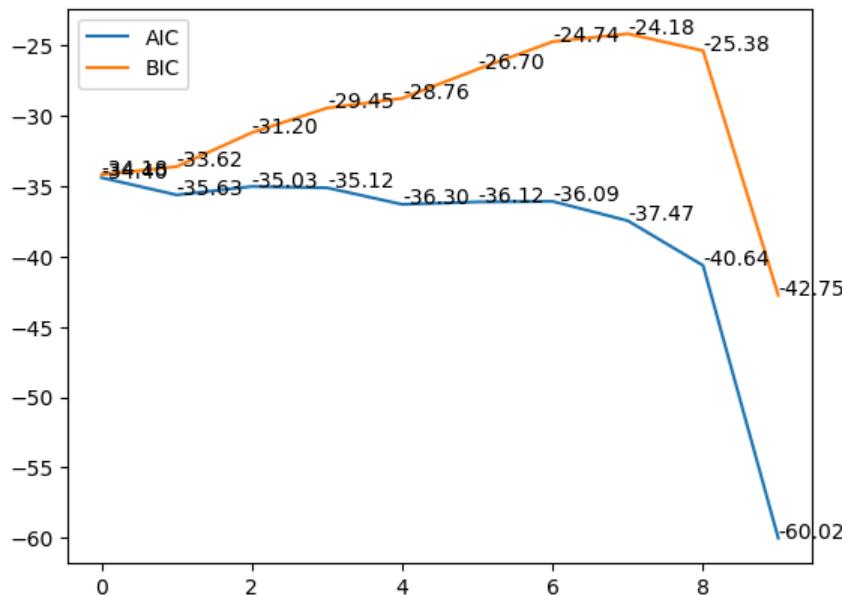
```
In [12]: from statsmodels.tsa.api import VAR

#Construct a model selection method to choose the optimal AIC or BIC value

def var_model_selection(data):
    lag_selection_space = range(0, 10)
    aic_best_lag = 0
    bic_best_lag = 0
    best_aic = 1000000
    best_bic = 1000000
    aic = []
    bic = []
    # Iterate over the predefined lag selection space
    for lag in lag_selection_space:
        print(f"Fitting VAR({lag})")
        # Fit the VAR model
        model = VAR(data)
        results = model.fit(lag)
        aic.append(results.aic)
        bic.append(results.bic)
    # If the current AIC or BIC value for the lag period is better than the previous value,
    # we can update the optimal lag period and the best AIC or BIC value.
    if results.aic < best_aic:
        best_aic = results.aic
        aic_best_lag = lag
    if results.bic < best_bic:
        best_bic = results.bic
        bic_best_lag = lag
    # Output the optimal lag period and its corresponding AIC and BIC values.
    print(f"Optimal lag selected by AIC: {aic_best_lag}, AIC = {best_aic}")
    print(f"Optimal lag selected by BIC: {bic_best_lag}, BIC = {best_bic}")
    # Plot the graph of AIC and BIC values as they vary with the lag period.
    plt.plot(lag_selection_space, aic, label="AIC")
    for x, y in zip(lag_selection_space, aic):
        plt.text(x, y, f"{y:.2f}")
    plt.plot(lag_selection_space, bic, label="BIC")
    for x, y in zip(lag_selection_space, bic):
        plt.text(x, y, f"{y:.2f}")
    plt.legend()
    plt.show()
```

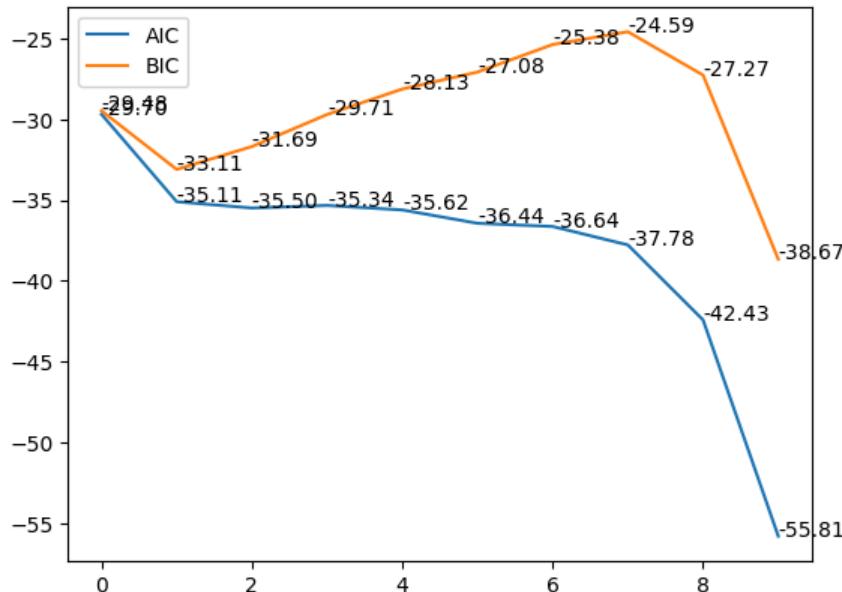
```
In [13]: # Apply model selection methods on the variables.  
var_model_selection(var_data1)
```

```
Fitting VAR(0)  
Fitting VAR(1)  
Fitting VAR(2)  
Fitting VAR(3)  
Fitting VAR(4)  
Fitting VAR(5)  
Fitting VAR(6)  
Fitting VAR(7)  
Fitting VAR(8)  
Fitting VAR(9)  
Lag selected by AIC: 9, AIC = -60.01708500023309  
Lag selected by BIC: 9, BIC = -42.75340220605155  
  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)  
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.  
    self._init_dates(dates, freq)
```



According to the AIC, we can choose local optimal p=4 for var1.

```
In [14]: var_model_selection(var_data2)
```



According to the AIC, we can choose local optimal p=1 for var2.

Fit the VAR model.

In [15]:

```

var1_lag=4
var1=VAR(var_data1)
var1_results = var1.fit(var1_lag)

#show the result
from IPython.display import display, Markdown
coefs = var1_results.params['Income inequality - Seasonal Adjusted Difference']
std_errors = var1_results.bse['Income inequality - Seasonal Adjusted Difference']
p_values = var1_results.pvalues['Income inequality - Seasonal Adjusted Difference']

results_df = pd.DataFrame({
    'Coefficient': coefs,
    'Standard Error': std_errors,
    'P-value': p_values
})
display(Markdown('### Result for Income inequality - Seasonal Adjusted Difference'))
display(results_df)

```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

Result for Income inequality - Seasonal Adjusted Difference

		Coefficient	Standard Error	P-value
	const	-0.001684	0.000675	0.012598
L1.Qualterly % Change of Inflation (qualterly) Difference		-0.002463	0.001883	0.190976
	L1.Real GDP Change	0.000538	0.000710	0.448510
	L1.Quarterly Nikkei 225 Index Change	-0.007471	0.006542	0.253473
	L1.% Quarterly Change- MB (SA) Difference	0.013649	0.018421	0.458725
	L1.Income inequality - Seasonal Adjusted Difference	-0.761863	0.186536	0.000044
	L1.Unemployment rate - Seasonal Adjusted Difference	0.004575	0.002569	0.074960
	L1.earthquake	-0.001204	0.004785	0.801281
	L1.tax	0.004733	0.005187	0.361446
L2.Qualterly % Change of Inflation (qualterly) Difference		0.000907	0.001902	0.633497
	L2.Real GDP Change	0.000880	0.000704	0.211027
	L2.Quarterly Nikkei 225 Index Change	-0.009403	0.007443	0.206432
	L2.% Quarterly Change- MB (SA) Difference	0.026545	0.019617	0.176002
	L2.Income inequality - Seasonal Adjusted Difference	-0.488424	0.205977	0.017728
	L2.Unemployment rate - Seasonal Adjusted Difference	0.004744	0.002578	0.065796
	L2.earthquake	0.005235	0.005895	0.374521
	L2.tax	0.005623	0.005133	0.273350
L3.Qualterly % Change of Inflation (qualterly) Difference		-0.000560	0.001840	0.760668
	L3.Real GDP Change	0.000616	0.000667	0.355531
	L3.Quarterly Nikkei 225 Index Change	-0.009629	0.007728	0.212737
	L3.% Quarterly Change- MB (SA) Difference	0.023965	0.019996	0.230731
	L3.Income inequality - Seasonal Adjusted Difference	-0.535670	0.200546	0.007561
	L3.Unemployment rate - Seasonal Adjusted Difference	0.002866	0.002618	0.273617
	L3.earthquake	-0.003015	0.006045	0.617892
	L3.tax	0.007959	0.005287	0.132191
L4.Qualterly % Change of Inflation (qualterly) Difference		-0.001162	0.001646	0.480172
	L4.Real GDP Change	0.001062	0.000714	0.136796
	L4.Quarterly Nikkei 225 Index Change	-0.001436	0.007489	0.847975
	L4.% Quarterly Change- MB (SA) Difference	0.007058	0.019935	0.723296
	L4.Income inequality - Seasonal Adjusted Difference	0.223535	0.175079	0.201685
	L4.Unemployment rate - Seasonal Adjusted Difference	0.003762	0.002656	0.156666
	L4.earthquake	0.001271	0.004647	0.784372
	L4.tax	0.004249	0.005115	0.406142

```
In [16]: var2_lag=1
var2=VAR(var_data2)
var2_results = var2.fit(var2_lag)

#show the result
from IPython.display import display, Markdown
coefs = var2_results.params['Income inequality - Seasonal Adjusted']
std_errors = var2_results.bse['Income inequality - Seasonal Adjusted']
p_values = var2_results.pvalues['Income inequality - Seasonal Adjusted']

results_df = pd.DataFrame({
    'Coefficient': coefs,
    'Standard Error': std_errors,
    'P-value': p_values
})
display(Markdown('### Result for Income inequality - Seasonal Adjusted'))
display(results_df)
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

Result for Income inequality - Seasonal Adjusted

	Coefficient	Standard Error	P-value
const	0.120978	0.023954	4.408681e-07
L1.Qualterly % Change of Inflation (qualterly)	0.000555	0.000771	4.714966e-01
L1.Real GDP Change	0.001288	0.000706	6.818708e-02
L1.Quarterly Nikkei 225 Index Change	-0.008404	0.007550	2.656675e-01
L1.% Quarterly Change- MB (SA)	0.025452	0.017652	1.493301e-01
L1.Income inequality - Seasonal Adjusted	0.048509	0.126098	7.004625e-01
L1.Unemployment rate - Seasonal Adjusted	0.002715	0.000532	3.309594e-07
L1.earthquake	0.000301	0.004483	9.463914e-01
L1.tax	0.000400	0.006205	9.485826e-01

Part A summary, Interpret the VAR results:

从VAR_1模型来看，对于“收入不平等的一阶差分”（经过季节性调整）这一变量而言，较为显著的预测变量为前一期的“收入不平等的一阶差分”（p=0.000044）与前一期的“失业率的一阶差分”（p=0.074960），以及前二期的“收入不平等的一阶差分”（p=0.017728）与前一期的“失业率的一阶差分”（p=0.065796）；以及前三期的“收入不平等的一阶差分”（p=0.007561）。这表明在此模型中，“收入不平等的一阶差分”的历史数据对于其当期值有显著的预测意义，同时之前的“失业率的一阶差分”也对于收入不平等具有较为显著的预测意义。然而其他变量例如通胀、GDP等宏观变量的历史信息对于“收入不平等的一阶差分”预测力度有限。

从VAR_2模型来看，对于“收入不平等”（经过季节性调整）这一变量而言，具有较大预测意义的是前一期的“实际GDP变化”（p=6.818708e-02）与前一期的“失业率”（p=3.309594e-07），然而这有可能受到未处理线性时间趋势带来的伪回归效应影响。

总的而言，对于收入不平等的预测能够较好地依赖于其自身的歷史信息以及失业率的历史信息。

Part B

```
In [17]: from statsmodels.tsa.vector_ar.svar_model import SVAR

A1 = np.asarray([[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 1]])
```

```
In [18]: #Fit SVAR on the variables we choose
svar1 = SVAR(var_data1, svar_type="A", A=A1)
svar1_results = svar1.fit(maxlags=var1_lag)
table = pd.DataFrame(svar1_results.A, columns=["inflation_diff_col", "gdp_change_col", "index_change_col", "mb_change_diff_col", "income_inequal_diff_col", "unemployment_diff_col", "earthquake_col", "tax_col"], index=["inflation_diff_col", "gdp_change_col", "index_change_col", "mb_change_diff_col", "income_inequal_diff_col", "unemployment_diff_col", "earthquake_col", "tax_col"])
table
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
warnings.warn("Maximum Likelihood optimization failed to "

Out[18]:

	inflation_diff_col	gdp_change_col	index_change_col	mb_change_diff_col	income_inequal_diff_col	unemployment_diff_col
inflation_diff_col	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
gdp_change_col	-0.819045	1.000000	0.000000	0.000000	0.000000	0.000000
index_change_col	0.097356	-0.027043	1.000000	0.000000	0.000000	0.000000
mb_change_diff_col	-0.007690	0.009545	-0.019697	1.000000	0.000000	0.000000
income_inequal_diff_col	0.000327	0.001419	-0.001178	-0.011064	1.000000	0.000000
unemployment_diff_col	-0.095081	0.038211	-0.059371	1.380528	-7.555124	0.000000
earthquake_col	0.022484	0.049340	-0.230622	-0.682513	-0.395556	0.000000
tax_col	-0.050972	0.021303	0.087127	-0.081797	0.127731	0.000000



```
In [19]: #Fit SVAR on the variables without difference process
svar2 = SVAR(var_data2, svar_type="A", A=A1)
svar2_results = svar2.fit(maxlags=var2_lag)
table = pd.DataFrame(svar2_results.A, columns=["inflation_col", "gdp_change_col", "index_change_col", "mb_change_col", "income_inequal_col", "unemployment_col", "earthquake_col", "tax_col"], index=[["inflation_col", "gdp_change_col", "index_change_col", "mb_change_col", "income_inequal_col", "unemployment_col", "earthquake_col", "tax_col"]])
table
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
warnings.warn("Maximum Likelihood optimization failed to "

Out[19]:

	inflation_col	gdp_change_col	index_change_col	mb_change_col	income_inequal_col	unemployment_col	earthquake_col
inflation_col	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
gdp_change_col	-0.066664	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
index_change_col	0.022323	-0.025683	1.000000	0.000000	0.000000	0.000000	0.000000
mb_change_col	-0.002646	0.004524	-0.030328	1.000000	0.000000	0.000000	0.000000
income_inequal_col	-0.000190	0.000947	-0.006281	-0.014988	1.000000	0.000000	0.000000
unemployment_col	0.042638	-0.001934	-0.067673	-0.995143	-8.791983	1.000000	0.000000
earthquake_col	-0.010929	0.048942	-0.089472	-1.351300	1.949420	0.009763	1.000000
tax_col	-0.036270	-0.001577	0.098057	-0.383806	2.162675	-0.028555	0.000000



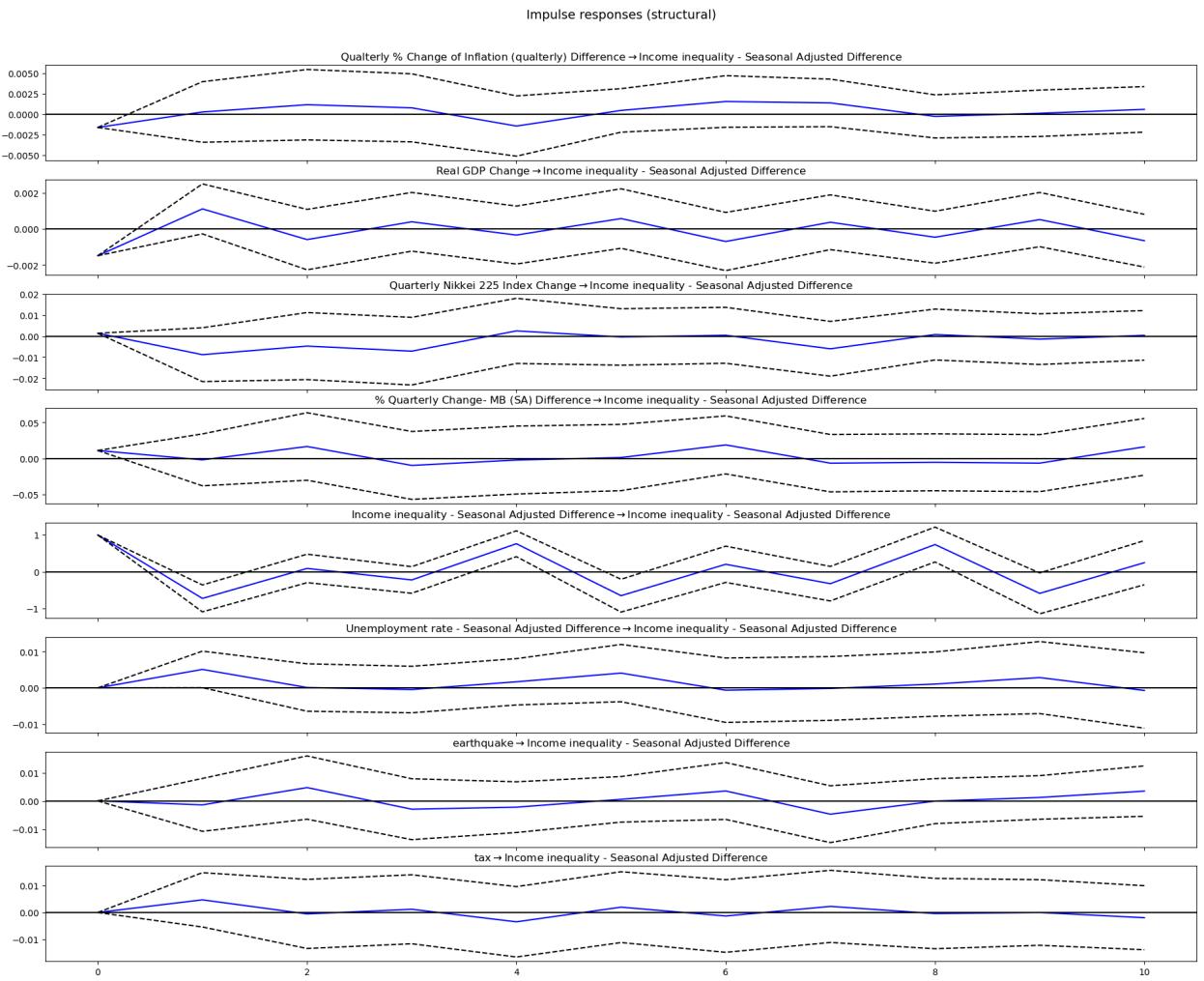
Part B summary, Why we need SVAR on top of VAR:

VAR模型的每个变量的误差项互相之间并不保证不相关，因此无法区分两个变量之间的关系是通过当期相互关联还是通过t对t+1,t+2...的传导的影响。SVAR模型使得变量之间的重构后误差项不再相关，可以一举两得，既通过协方差矩阵捕捉到变量之间同期的关联，也可以单独干净地看到一个变量的外生冲击如何通过之后期进行传导对于另一些变量产生影响，也即可以作正交脉冲响应分析。

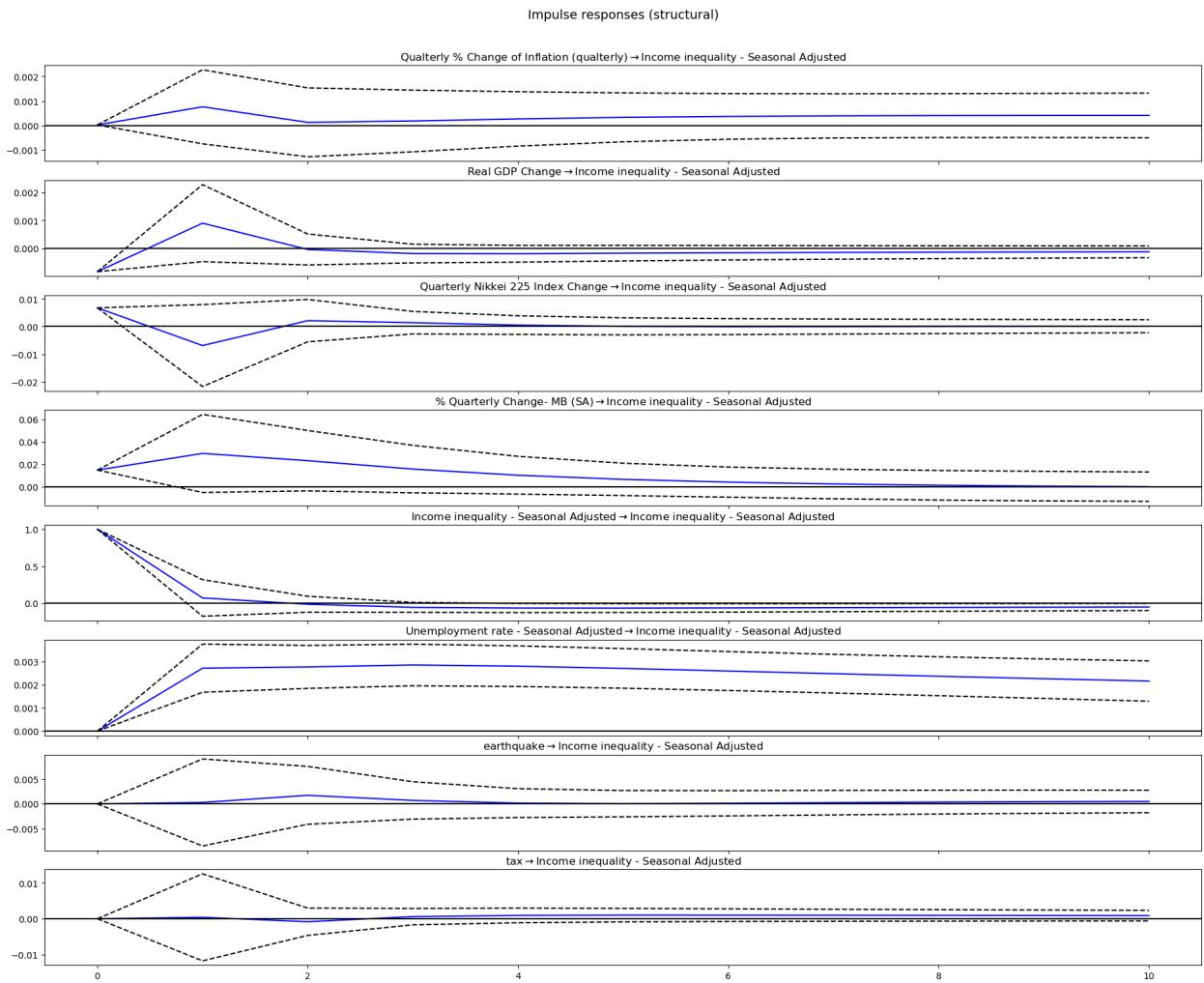
上面显示的A矩阵表示了各个宏观变量之间的当期影响系数。

Part C

```
In [20]: irf1 = svar1_results.irf(10)
plot1 = irf1.plot(response='income_inequal_diff_col', figsize=(20, 16))
```



```
In [21]: irf2 = svar2_results.irf(10)
plot2 = irf2.plot(response='income_inequal_sa_col', figsize=(20, 16))
```



Part C summary, impulse response plots:

上面图显示了正交脉冲响应图。

如图所示，对于“收入不平等”或者“收入不平等的一阶差分”，从模型1而言，难以看到其他变量冲击对“收入不平等一阶差分”的影响。

从模型2而言，可见“失业率”的冲击可以对“收入不平等”产生非常持续的显著的影响。同时“量宽MB”冲击可能有一期的影响（不是很显著），但是并不持续。

Part D

The ARDL model

First fit the ARDL model with series of I(1) and I(0).

```
In [22]: from statsmodels.tsa.api import ARDL

var_data2 = df[[inflation_col, gdp_change_col, index_col, index_change_col, mb_change_col, income_inequal_sa_col, \
                unemployment_sa_col, earthquake_col, tax_col]]
ardl = ARDL(var_data2[income_inequal_sa_col], 1, var_data2[[inflation_col, gdp_change_col, index_change_col, mb_change_col, \
                unemployment_sa_col, earthquake_col, tax_col]], 1, seasonal=True, trend="ct", period=4)
ardl_results = ardl.fit()
ardl_results.summary()

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and
d will be ignored when e.g. forecasting.
    self._init_dates(dates, freq)
```

Out[22]: ARDL Model Results

Dep. Variable:	Income inequality - Seasonal Adjusted	No. Observations:	91			
Model:	Seas. ARDL(1, 1, 1, 1, 1, 1, 1)	Log Likelihood	349.688			
Method:	Conditional MLE	S.D. of innovations	0.005			
Date:	Sat, 01 Jun 2024	AIC	-657.376			
Time:	21:16:12	BIC	-604.880			
Sample:	1	HQIC	-636.206			
	91					
	coef	std err	z	P> z	[0.025	0.975]
const	0.0467	0.030	1.553	0.125	-0.013	0.107
trend	-0.0002	4.47e-05	-3.372	0.001	-0.000	-6.16e-05
s(2,4)	0.0003	0.002	0.155	0.877	-0.003	0.004
s(3,4)	-0.0004	0.002	-0.232	0.818	-0.004	0.003
s(4,4)	0.0001	0.002	0.072	0.943	-0.003	0.004
Income inequality - Seasonal Adjusted.L1	0.0563	0.156	0.362	0.718	-0.254	0.366
Quarterly % Change of Inflation (quarterly).L0	0.0007	0.001	0.469	0.640	-0.002	0.004
Quarterly % Change of Inflation (quarterly).L1	-0.0006	0.002	-0.422	0.674	-0.004	0.002
Real GDP Change.L0	-0.0012	0.001	-1.574	0.120	-0.003	0.000
Real GDP Change.L1	0.0011	0.001	1.531	0.130	-0.000	0.002
Quarterly Nikkei 225 Index Change.L0	0.0057	0.008	0.734	0.466	-0.010	0.021
Quarterly Nikkei 225 Index Change.L1	-0.0024	0.008	-0.287	0.775	-0.019	0.014
% Quarterly Change- MB (SA).L0	0.0344	0.024	1.435	0.156	-0.013	0.082
% Quarterly Change- MB (SA).L1	0.0241	0.024	0.994	0.323	-0.024	0.072
Unemployment rate - Seasonal Adjusted.L0	0.0019	0.003	0.737	0.464	-0.003	0.007
Unemployment rate - Seasonal Adjusted.L1	0.0027	0.003	0.891	0.376	-0.003	0.009
earthquake.L0	-0.0062	0.005	-1.142	0.258	-0.017	0.005
earthquake.L1	0.0035	0.005	0.638	0.525	-0.007	0.014
tax.L0	-0.0046	0.006	-0.750	0.456	-0.017	0.008
tax.L1	-0.0016	0.006	-0.258	0.797	-0.014	0.011

Second fit the ARDL model only with series of I(1) without I(0).

```
In [23]: ardl = ARDL(var_data2[income_inequal_sa_col], 1, var_data2[[mb_change_col,unemployment_sa_col]], 1, seasonal=True, trend=True)
ardl_results = ardl.fit()
ardl_results.summary()
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided an d will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

Out[23]: ARDL Model Results

Dep. Variable:	Income inequality - Seasonal Adjusted	No. Observations:	91			
Model:	Seas. ARDL(1, 1, 1)	Log Likelihood	345.212			
Method:	Conditional MLE	S.D. of innovations	0.005			
Date:	Sat, 01 Jun 2024	AIC	-668.424			
Time:	21:16:15	BIC	-640.926			
Sample:	1	HQIC	-657.335			
	91					
	coef	std err	z	P> z	[0.025	0.975]
const	0.0585	0.028	2.060	0.043	0.002	0.115
trend	-0.0002	4.32e-05	-3.625	0.001	-0.000	-7.06e-05
s(2,4)	0.0007	0.002	0.420	0.675	-0.003	0.004
s(3,4)	-0.0001	0.002	-0.082	0.935	-0.003	0.003
s(4,4)	0.0006	0.002	0.341	0.734	-0.003	0.004
Income inequality - Seasonal Adjusted.L1	-0.0290	0.145	-0.200	0.842	-0.317	0.259
% Quarterly Change- MB (SA).L0	0.0328	0.022	1.492	0.140	-0.011	0.077
% Quarterly Change- MB (SA).L1	0.0239	0.022	1.093	0.278	-0.020	0.067
Unemployment rate - Seasonal Adjusted.L0	0.0012	0.002	0.487	0.627	-0.004	0.006
Unemployment rate - Seasonal Adjusted.L1	0.0036	0.003	1.262	0.211	-0.002	0.009

Part D summary, fit the ARDL:

上面的表结果显示了使用I(1)和I(0)序列以及只用I(1)序列进行的ARDL模型结果。

然而根据模型的估计,我并没有得到很显著的系数。

Part E

First we use Engle Granger Test

```
In [236]: from arch.unitroot import engle_granger
test1 = engle_granger(var_data2[income_inequal_sa_col], var_data2[[mb_change_col,unemployment_sa_col]], trend="ct")
test1
```

Out[236]: Engle-Granger Cointegration Test

Test Statistic	-2.183
P-value	0.827
ADF Lag length	3
Estimated Root p (y+1)	0.660

Trend: Constant

Critical Values: -3.96 (10%), -4.28 (5%), -4.91 (1%)

Null Hypothesis: No Cointegration

Alternative Hypothesis: Cointegration

Distribution Order: 2

Second we use ARDL Test

```
In [24]: from statsmodels.tsa.ardl import BoundsTestResult
from statsmodels.tsa.ardl import UECM
```

use I(0) and I(1)

```
In [25]: var_data2 = df[[inflation_col,gdp_change_col,index_col,index_change_col,mb_change_col,income_inequal_sa_col,\n                    unemployment_sa_col,earthquake_col,tax_col]]\nuecm = UECM(var_data2[income_inequal_sa_col], 1, var_data2[[inflation_col,gdp_change_col,index_change_col,mb_change_col,\n                    unemployment_sa_col,earthquake_col,tax_col]], 1, trend='ct',\n                    period=4)\nuecm_results = uecm.fit()\nuecm_results.summary()\nboundsTestResult = uecm_results.bounds_test(case=1)\nboundsTestResult
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

```
Out[25]: BoundsTestResult\nStat: 2.33805\nUpper P-value: 0.232\nLower P-value: 0.014\nNull: No Cointegration\nAlternative: Possible Cointegration
```

use only I(1)

```
In [26]: var_data2 = df[[inflation_col,gdp_change_col,index_col,index_change_col,mb_change_col,income_inequal_sa_col,\n                    unemployment_sa_col,earthquake_col,tax_col]]\nuecm = UECM(var_data2[income_inequal_sa_col], 1, var_data2[[index_col,mb_change_col,unemployment_sa_col]]\\,\n            1, trend='ct', seasonal=True, period=4)\nuecm_results = uecm.fit()\nuecm_results.summary()\nboundsTestResult = uecm_results.bounds_test(case=1)\nboundsTestResult
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

```
Out[26]: BoundsTestResult\nStat: 5.37683\nUpper P-value: 0.00186\nLower P-value: 8.52e-05\nNull: No Cointegration\nAlternative: Possible Cointegration
```

Part E summary, test the ways of cointegration among those I(1) sequences:

上面的结果显示了I(1)序列的Engle Granger Test与ARDL Test(bound test)

前者的P值为0.827,显示不存在协整关系.

然而ARDL Test(bound test)的Upper bound的p值为0.00186,显示可能存在协整关系.

Part F

```
In [27]: from statsmodels.tsa.vector_ar import vecm
from statsmodels.tsa.vector_ar.vecm import coint_johansen
# Select the order of the VECM
vecm_lag = vecm.select_order(data=var_data2[['income_inequal_sa_col', 'mb_change_col', 'unemployment_sa_col']], maxlags=10)
# Display the result
vecm_lag.summary()
```

D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
self._init_dates(dates, freq)

Out[27]: VECM Order Selection (* highlights the minimums)

	AIC	BIC	FPE	HQIC
0	-20.15	-19.79*	1.774e-09	-20.01
1	-20.28	-19.66	1.554e-09	-20.03
2	-20.24	-19.35	1.625e-09	-19.88
3	-20.43	-19.26	1.358e-09	-19.96
4	-21.04*	-19.61	7.409e-10*	-20.47*
5	-20.88	-19.18	8.805e-10	-20.20
6	-20.88	-18.91	8.970e-10	-20.09
7	-20.96	-18.72	8.456e-10	-20.06
8	-20.86	-18.36	9.535e-10	-19.86
9	-20.78	-18.01	1.076e-09	-19.67
10	-20.65	-17.61	1.281e-09	-19.43

```
In [28]: # Perform the Johansen cointegration rank test
from statsmodels.tsa.vector_ar.vecm import coint_johansen
coint_result = coint_johansen(var_data2[[income_inequal_sa_col,mb_change_col,unemployment_sa_col]], 1, 4)
#Trace Statistics
print("Trace Statistics:")
print(coint_result.lrl)
#Maximum Eigenvalue Statistic
print("Maximum Eigenvalue Statistic:")
print(coint_result.lr2)
# From the statistics compared to the critical value(not shown), we believe that there are 2 cointegration relationships e
print("Trace Statistics:", coint_result.lrl)
print("Critical Values (90%, 95%, 99%):", coint_result.cvt)
print("Eigenvalues:", coint_result.eig)
print("Maximum Eigenvalue Statistics:", coint_result.lr2)
print("Critical Values for max test (90%, 95%, 99%):", coint_result.cvm)
```

Trace Statistics:
[38.63210302 17.17009736 3.35936925]
Maximum Eigenvalue Statistic:
[21.46200565 13.81072811 3.35936925]
Trace Statistics: [38.63210302 17.17009736 3.35936925]
Critical Values (90%, 95%, 99%): [[32.0645 35.0116 41.0815]
[16.1619 18.3985 23.1485]
[2.7055 3.8415 6.6349]]
Eigenvalues: [0.22085507 0.14835871 0.03830933]
Maximum Eigenvalue Statistics: [21.46200565 13.81072811 3.35936925]
Critical Values for max test (90%, 95%, 99%): [[21.8731 24.2522 29.2631]
[15.0006 17.1481 21.7465]
[2.7055 3.8415 6.6349]]

We use r=2 according to Trace Statistics at Critical Values of 90%, and we can also try r=1 according to Trace Statistics at Critical Values of 95%.

```
In [29]: #Fit vecm model use lag=1 and rank=2
vecm_lag = 4
coint_rank = 2
vecm_model = vecm.VECM(var_data2[['income_inequal_sa_col',mb_change_col,unemployment_sa_col]], k_ar_diff=vecm_lag, coint_ran
vecm_results = vecm_model.fit()
vecm_results.summary()
```

```
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided an
d will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

Out[29]:

Det. terms outside the coint. relation & lagged endog. parameters for equation Income inequality -
Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	-0.5584	0.143	-3.914	0.000	-0.838	-0.279
L1.% Quarterly Change- MB (SA)	0.0008	0.017	0.048	0.961	-0.032	0.034
L1.Unemployment rate - Seasonal Adjusted	0.0043	0.002	2.327	0.020	0.001	0.008
L2.Income inequality - Seasonal Adjusted	-0.3149	0.153	-2.055	0.040	-0.615	-0.015
L2.% Quarterly Change- MB (SA)	0.0103	0.017	0.622	0.534	-0.022	0.043
L2.Unemployment rate - Seasonal Adjusted	0.0037	0.002	1.886	0.059	-0.000	0.008
L3.Income inequality - Seasonal Adjusted	-0.4575	0.151	-3.038	0.002	-0.753	-0.162
L3.% Quarterly Change- MB (SA)	0.0019	0.016	0.119	0.905	-0.029	0.032
L3.Unemployment rate - Seasonal Adjusted	0.0020	0.002	1.018	0.309	-0.002	0.006
L4.Income inequality - Seasonal Adjusted	0.2292	0.139	1.651	0.099	-0.043	0.501
L4.% Quarterly Change- MB (SA)	-0.0125	0.015	-0.844	0.399	-0.041	0.017
L4.Unemployment rate - Seasonal Adjusted	0.0031	0.002	1.594	0.111	-0.001	0.007

Det. terms outside the coint. relation & lagged endog. parameters for equation % Quarterly Change-
MB (SA)

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	2.7415	0.986	2.781	0.005	0.809	4.674
L1.% Quarterly Change- MB (SA)	-0.1270	0.116	-1.091	0.275	-0.355	0.101
L1.Unemployment rate - Seasonal Adjusted	-0.0317	0.013	-2.453	0.014	-0.057	-0.006
L2.Income inequality - Seasonal Adjusted	2.5149	1.059	2.375	0.018	0.440	4.590
L2.% Quarterly Change- MB (SA)	-0.1939	0.115	-1.689	0.091	-0.419	0.031
L2.Unemployment rate - Seasonal Adjusted	-0.0164	0.013	-1.220	0.222	-0.043	0.010
L3.Income inequality - Seasonal Adjusted	2.1238	1.040	2.041	0.041	0.085	4.163
L3.% Quarterly Change- MB (SA)	-0.1412	0.107	-1.318	0.188	-0.351	0.069
L3.Unemployment rate - Seasonal Adjusted	-0.0090	0.014	-0.652	0.515	-0.036	0.018
L4.Income inequality - Seasonal Adjusted	2.6028	0.959	2.714	0.007	0.723	4.483
L4.% Quarterly Change- MB (SA)	-0.1501	0.102	-1.469	0.142	-0.350	0.050
L4.Unemployment rate - Seasonal Adjusted	0.0089	0.014	0.655	0.512	-0.018	0.036

Det. terms outside the coint. relation & lagged endog. parameters for equation Unemployment rate -
Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	-4.2978	6.944	-0.619	0.536	-17.908	9.313
L1.% Quarterly Change- MB (SA)	2.0177	0.820	2.460	0.014	0.410	3.625
L1.Unemployment rate - Seasonal Adjusted	-0.0124	0.091	-0.136	0.892	-0.191	0.166
L2.Income inequality - Seasonal Adjusted	3.2212	7.457	0.432	0.666	-11.394	17.836
L2.% Quarterly Change- MB (SA)	0.9834	0.809	1.216	0.224	-0.601	2.568
L2.Unemployment rate - Seasonal Adjusted	-0.0261	0.095	-0.275	0.783	-0.212	0.160
L3.Income inequality - Seasonal Adjusted	8.5542	7.328	1.167	0.243	-5.809	22.918
L3.% Quarterly Change- MB (SA)	1.4972	0.755	1.984	0.047	0.018	2.976
L3.Unemployment rate - Seasonal Adjusted	0.0427	0.097	0.440	0.660	-0.147	0.233
L4.Income inequality - Seasonal Adjusted	12.3515	6.755	1.828	0.067	-0.889	25.592
L4.% Quarterly Change- MB (SA)	0.7132	0.720	0.991	0.322	-0.698	2.124
L4.Unemployment rate - Seasonal Adjusted	0.5117	0.096	5.345	0.000	0.324	0.699

Loading coefficients (alpha) for equation Income inequality
- Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
ec1	-0.1953	0.065	-3.007	0.003	-0.323	-0.068
ec2	0.0082	0.014	0.597	0.550	-0.019	0.035

Loading coefficients (alpha) for equation % Quarterly
Change- MB (SA)

	coef	std err	z	P> z	[0.025	0.975]
ec1	0.0922	0.449	0.205	0.837	-0.788	0.972
ec2	-0.2672	0.095	-2.799	0.005	-0.454	-0.080

Loading coefficients (alpha) for equation Unemployment
rate - Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
ec1	2.0951	3.161	0.663	0.508	-4.101	8.291
ec2	-0.9101	0.672	-1.353	0.176	-2.228	0.408

Cointegration relations for loading-coefficients-column 1

	coef	std err	z	P> z	[0.025	0.975]
beta.1	1.0000		0	0	0.000	1.000
beta.2		0	0	0	0.000	0
beta.3	-0.0058	5.31e-05	-109.993	0.000	-0.006	-0.006

Cointegration relations for loading-coefficients-column 2

	coef	std err	z	P> z	[0.025	0.975]
beta.1	0	0	0	0.000	0	0
beta.2	1.0000		0	0	0.000	1.000
beta.3	-0.0007	0.000	-2.975	0.003	-0.001	-0.000

```
In [30]: #Fit vecm model use lag=1 and rank=1
vecm_lag = 4
coint_rank = 1
vecm_model = vecm.VECM(var_data2[['income_inequal_sa_col',mb_change_col,unemployment_sa_col]], k_ar_diff=vecm_lag, coint_ran
vecm_results = vecm_model.fit()
vecm_results.summary()
```

```
D:\anaconda\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: An unsupported index was provided an
d will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

Out[30]:

Det. terms outside the coint. relation & lagged endog. parameters for equation Income inequality -
Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	-0.6118	0.143	-4.281	0.000	-0.892	-0.332
L1.% Quarterly Change- MB (SA)	-0.0107	0.016	-0.665	0.506	-0.042	0.021
L1.Unemployment rate - Seasonal Adjusted	0.0047	0.002	2.472	0.013	0.001	0.008
L2.Income inequality - Seasonal Adjusted	-0.3405	0.156	-2.185	0.029	-0.646	-0.035
L2.% Quarterly Change- MB (SA)	0.0008	0.016	0.049	0.961	-0.031	0.033
L2.Unemployment rate - Seasonal Adjusted	0.0038	0.002	1.895	0.058	-0.000	0.008
L3.Income inequality - Seasonal Adjusted	-0.4737	0.154	-3.086	0.002	-0.775	-0.173
L3.% Quarterly Change- MB (SA)	-0.0049	0.015	-0.318	0.751	-0.035	0.025
L3.Unemployment rate - Seasonal Adjusted	0.0020	0.002	1.000	0.318	-0.002	0.006
L4.Income inequality - Seasonal Adjusted	0.2322	0.142	1.639	0.101	-0.046	0.510
L4.% Quarterly Change- MB (SA)	-0.0177	0.015	-1.189	0.235	-0.047	0.011
L4.Unemployment rate - Seasonal Adjusted	0.0032	0.002	1.580	0.114	-0.001	0.007

Det. terms outside the coint. relation & lagged endog. parameters for equation % Quarterly Change-
MB (SA)

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	2.3771	0.987	2.408	0.016	0.443	4.312
L1.% Quarterly Change- MB (SA)	-0.2056	0.111	-1.852	0.064	-0.423	0.012
L1.Unemployment rate - Seasonal Adjusted	-0.0293	0.013	-2.235	0.025	-0.055	-0.004
L2.Income inequality - Seasonal Adjusted	2.3399	1.076	2.174	0.030	0.230	4.450
L2.% Quarterly Change- MB (SA)	-0.2590	0.112	-2.317	0.021	-0.478	-0.040
L2.Unemployment rate - Seasonal Adjusted	-0.0158	0.014	-1.148	0.251	-0.043	0.011
L3.Income inequality - Seasonal Adjusted	2.0127	1.060	1.898	0.058	-0.065	4.091
L3.% Quarterly Change- MB (SA)	-0.1873	0.106	-1.759	0.079	-0.396	0.021
L3.Unemployment rate - Seasonal Adjusted	-0.0089	0.014	-0.636	0.525	-0.036	0.019
L4.Income inequality - Seasonal Adjusted	2.6235	0.979	2.680	0.007	0.705	4.542
L4.% Quarterly Change- MB (SA)	-0.1854	0.103	-1.808	0.071	-0.386	0.016
L4.Unemployment rate - Seasonal Adjusted	0.0092	0.014	0.661	0.509	-0.018	0.036

Det. terms outside the coint. relation & lagged endog. parameters for equation Unemployment rate -
Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
L1.Income inequality - Seasonal Adjusted	-4.9412	6.820	-0.725	0.469	-18.308	8.425
L1.% Quarterly Change- MB (SA)	1.8790	0.767	2.448	0.014	0.375	3.383
L1.Unemployment rate - Seasonal Adjusted	-0.0082	0.091	-0.091	0.928	-0.186	0.169
L2.Income inequality - Seasonal Adjusted	2.9122	7.438	0.392	0.695	-11.666	17.490
L2.% Quarterly Change- MB (SA)	0.8684	0.773	1.124	0.261	-0.646	2.382
L2.Unemployment rate - Seasonal Adjusted	-0.0250	0.095	-0.263	0.793	-0.211	0.161
L3.Income inequality - Seasonal Adjusted	8.3581	7.326	1.141	0.254	-6.001	22.717
L3.% Quarterly Change- MB (SA)	1.4158	0.736	1.924	0.054	-0.026	2.858
L3.Unemployment rate - Seasonal Adjusted	0.0427	0.097	0.440	0.660	-0.148	0.233
L4.Income inequality - Seasonal Adjusted	12.3880	6.764	1.831	0.067	-0.869	25.645
L4.% Quarterly Change- MB (SA)	0.6509	0.709	0.918	0.358	-0.738	2.040
L4.Unemployment rate - Seasonal Adjusted	0.5122	0.096	5.343	0.000	0.324	0.700

Loading coefficients (alpha) for equation Income inequality
- Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
ec1	-0.1099	0.048	-2.277	0.023	-0.205	-0.015

Loading coefficients (alpha) for equation % Quarterly

Change- MB (SA)

	coef	std err	z	P> z	[0.025	0.975]
ec1	0.6753	0.334	2.025	0.043	0.022	1.329

Loading coefficients (alpha) for equation Unemployment
rate - Seasonal Adjusted

	coef	std err	z	P> z	[0.025	0.975]
ec1	3.1246	2.305	1.356	0.175	-1.392	7.641

Cointegration relations for loading-coefficients-column 1

	coef	std err	z	P> z	[0.025	0.975]
beta.1	1.0000	0	0	0.000	1.000	1.000
beta.2	-0.2268	0.073	-3.107	0.002	-0.370	-0.084
beta.3	-0.0057	7.54e-05	-75.303	0.000	-0.006	-0.006

Part F summary:

上面的Trace Statistics显示变量之间存在1种(95%显著性水平)或者2种(90%显著性水平)协整关系。

同时上面的表结果Cointegration relations for loading-coefficients展示了协整系数,即表明三个变量之间存在着长期关系,

rank=2的beta系数分别为(1.0000 -3.298e-17 -0.0058)和(2.316e-16 1.0000 -0.0007) rank=1的beta系数分别为(1.0000 -0.2268 -0.0057)

且这些协整系数都显著。

以rank=1为例子,三个变量的长期均衡关系可以表示为income_inequal=(-0.2268)mb_change+(-0.0057)unemployment_rate+常数项/误差项

END