

Neural Networks and Deep Learning Project 1

(Cheng Zhong 16307110259)

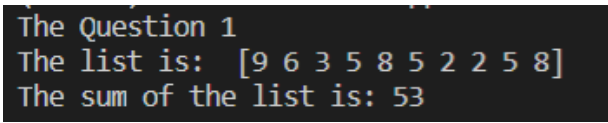
Preparation

```
import random
import numpy as np
import torch
from torch.autograd import Variable
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVC
from torchvision import transforms
import time
import pickle
from PIL import Image
import math
import re
```

1 Write a Python function to sum all the numbers in a list.

```
def Q1():
    a_list = np.random.randint(10,size=10)
    print('The Question 1')
    print('The list is: ', a_list)
    print('The sum of the list is:', sum(a_list))
    return 0
```

The result of Question 1 is:

A screenshot of a terminal window showing the output of the Q1 function. The text is as follows:

```
The Question 1
The list is: [9 6 3 5 8 5 2 2 5 8]
The sum of the list is: 53
```

Figure 1: Result of Question 1

2 Write a Python function that takes a list and returns a new list with unique elements of the first list.

```
def Q2():
    a_list = np.random.randint(10,size=10)
    print('The Question 2')
    print('The list is: ', a_list)
    print('The new list is:', list(set(a_list)))
    return 0
```

The result of Question 2 is:

```
The Question 2
The list is: [5 4 0 0 8 6 1 6 0 7]
The new list is: [0, 1, 4, 5, 6, 7, 8]
```

Figure 2: Result of Question 2

3 Write a Python function that checks whether a passed string is palindrome or not. A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam or nurses run.

```
def Q3():
    print('The Question 3')
    string = input('Please input a string: ')
    string = string.replace(' ', '')
    if string[::-1] == string:
        print('It\'s a palindrome.')
    else:
        print('It\'s not a palindrome.')
```

The result of Question 3 is:

```
(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/python.exe d:/Documents/课程介绍/神经网络与深度学习/Project1/Project1.py
The Question 3
Please input a string: madam
It's a palindrome.

(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/python.exe d:/Documents/课程介绍/神经网络与深度学习/Project1/Project1.py
The Question 3
Please input a string: nurses run
It's a palindrome.

(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/python.exe d:/Documents/课程介绍/神经网络与深度学习/Project1/Project1.py
The Question 3
Please input a string: ChengZhong
It's not a palindrome.
```

Figure 3: Result of Question 3

4 Write a NumPy program to find the real and imaginary parts of an array of complex numbers.

```
def Q4():
    print('The Question 4')
    x = np.array([1+0j, 0.707+0.707j])
    print(x)
    solu = np.empty((x.shape[0],2))
    for i in range(x.shape[0]):
        solu[i] = [np.real(x[i]), np.imag(x[i])]
    print(solu)
```

The result of Question 4 is:

5 Write a Python program to add two binary numbers.

```
def Q5():
    print('The Question 5')
```

```
The Question 4
[[1. +0.j 0.707+0.707j]
 [1. 0.]
 [0.707 0.707]]
```

Figure 4: Result of Question 4

```
a = input('Please input 2 binary number: ').split(',')
maxl = max(len(a[0]), len(a[1]))
for i in range(2):
    a[i] = '0'*(maxl - len(a[i])) + a[i]
solution = ''
enter = 0
for i in range(maxl-1, -1, -1):
    tot = enter + int(a[0][i]) + int(a[1][i])
    if tot == 0:
        solution = '0' + solution
    elif tot == 1:
        solution = '1' + solution
    elif tot == 2:
        solution = '0' + solution
        enter = 1
    elif tot == 3:
        solution = '1' + solution
        enter = 1
if enter == 1:
    solution = '1' + solution
print('The solution is ', solution)
```

The result of Question 5 is:

```
The Question 5
Please input 2 binary number: 11,1
The solution is 100
```

Figure 5: Result of Question 5

6 You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

```
def Q6():
    print('The Question 6')
    # Define the ListNode class
    class ListNode:
        def __init__(self, x):
            self.val = x
            self.next = None

    def linkstore(link):
```

```
# Store the link
l = link.split('>')
head = ListNode(int(l[0]))
point = head
for i in range(1,len(l)):
    new = ListNode(int(l[i]))
    point.next = new
    point = point.next
return head

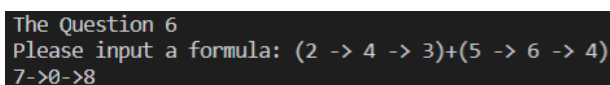
def formula2link(formula):
    # Transfer the formula to link node
    formula = formula.split('+')
    assert len(formula) == 2, 'Not a right formula'
    head1 = linkstore(formula[0].rstrip(')').lstrip('('))
    head2 = linkstore(formula[1].rstrip(')').lstrip('('))
    return head1, head2

def callinkplus(head1, head2):
    # plus the two link
    solution = ListNode(0)
    p_solu = solution
    carry_bit = 0
    while head1 or head2:
        x1 = head1 if head1 else 0
        x2 = head2 if head2 else 0
        solu = x1.val + x2.val + carry_bit
        carry_bit = solu // 10 # the plus calculation
        new = ListNode(solu%10)
        p_solu.next = new
        p_solu = p_solu.next
        head1 = head1.next
        head2 = head2.next
    return solution

def printlink(link):
    # print the link
    point = link.next
    while point.next:
        print(point.val, end = '>')
        point = point.next
    print(point.val)
    return 0

formula = input('Please input a formula: ')
head1, head2 = formula2link(formula)
solu1 = callinkplus(head1, head2)
printlink(solu1)
```

The result of Question 6 is:



```
The Question 6
Please input a formula: (2 -> 4 -> 3)+(5 -> 6 -> 4)
7->0->8
```

Figure 6: Result of Question 6

7 Implement bubble sort.

```
def Q7():
    print('The Question 7')
    # Initialize
    a_list = np.random.randint(10,size=10)

    def bubblesort(sort_list):
        if len(sort_list) < 2:
            return sort_list
        for i in range(len(sort_list)):
            for j in range(i+1, len(sort_list)):
                if sort_list[i] > sort_list[j]:
                    temp = sort_list[i]
                    sort_list[i] = sort_list[j]
                    sort_list[j] = temp

        return sort_list
    print('The raw data is: ', a_list)
    print('The solution is: ', bubblesort(a_list))
```

8 Implement bubble sort.

```
def Q8():
    print('The Question 8')
    a_list = [random.randint(1, 50) for i in range(10)]
    print('The raw data is: ', a_list)
    def merge(left, right):
        solu = []
        while left and right:
            if left[0] < right[0]:
                solu.append(left.pop(0))
            else:
                solu.append(right.pop(0))
        if left:
            solu += left
        if right:
            solu += right
        return solu

    def mergesort(sort_list):
        if len(sort_list) < 2:
            return sort_list
        mid = len(sort_list)//2
        left = sort_list[:mid:]
        right = sort_list[mid::]
        return merge(mergesort(left),mergesort(right))

    print('The solution is: ', mergesort(a_list))
```

9 Implement quick sort .

```
def Q9():
    print('The Question 9')
    a_list = np.random.randint(10,size=10)

    def quicksort(sort_list):
        if len(sort_list) < 2:
            return sort_list
        index = sort_list[0]
        left = []
        right = []
        for i in range(1,len(sort_list)):
            if sort_list[i] < index:
                left.append(sort_list[i])
            else:
                right.append(sort_list[i])
        return quicksort(left) + [index] + quicksort(right)
    print('The raw data is: ', a_list)
    print('The solution is: ', quicksort(a_list))
```

10 Implement shell sort .

```
def Q10():
    print('The Question 10')
    a_list = np.random.randint(10,size=20)

    def shellsort(sort_list):
        length = len(sort_list)
        gap = length // 2
        while gap > 0:
            for i in range(gap, length):
                for j in range(i, 0, -gap):
                    if sort_list[j] < sort_list[j-gap]:
                        temp = sort_list[j]
                        sort_list[j] = sort_list[j-gap]
                        sort_list[j-gap] = temp
            gap //= 2
        return sort_list
    print('The raw data is: ', a_list)
    print('The solution is: ', shellsort(a_list))
```

The result of Question 7-10 is:

11 Implement linear regression model and use autograd to optimize it by Pytorch.

```
def Q11():
    print('The Question 11')
    # Using a linear layer to calculate the linearregression
    class LinearRegression(nn.Module):
        def __init__(self, input_size = 1, output_size = 1):
            super(LinearRegression, self).__init__()
            self.linear = nn.Linear(input_size, output_size)
```

```

The Question 7
The raw data is: [0 0 2 4 5 1 0 2 0 6]
The solution is: [0 0 0 0 1 2 2 4 5 6]
The Question 8
The raw data is: [34, 2, 32, 30, 7, 23, 11, 29, 7, 7]
The solution is: [2, 7, 7, 7, 11, 23, 29, 30, 32, 34]
The Question 9
The raw data is: [4 0 7 0 5 6 6 9 7 5]
The solution is: [0, 0, 4, 5, 5, 6, 6, 7, 7, 9]
The Question 10
The raw data is: [6 7 7 5 3 6 5 6 9 6 3 6 8 0 0 1 9 8 9 2]
The solution is: [0 0 1 2 3 3 5 5 6 6 6 6 6 7 7 8 8 9 9 9]

```

Figure 7: Result of Question 7-10

```

def forward(self, x):
    return self.linear(x)
# Initialize the data
x = [3, 4, 5, 6, 7, 8, 9]
y = []
xx = np.linspace(2,10)
x_d = np.array(x)
for i in x:
    y.append(i * 2 + random.normalvariate(0,1)) # Add noise

iteration = 500
x = torch.FloatTensor(x)
x = x.view(-1,1)
x = Variable(x, requires_grad = True)

xx = torch.FloatTensor(xx)
xx = xx.view(-1,1)
xx = Variable(xx)

y = torch.FloatTensor(y)
y = y.view(-1,1)
y = Variable(y)

#Set the parameter of the model
input_size = 1
output_size = 1
model = LinearRegression(input_size, output_size)
# The Loss function
MSELoss = nn.MSELoss()

learning_rate = 1e-2
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
# Do the training
for i in range(iteration):
    optimizer.zero_grad()
    y_pred = model(x)
    loss = MSELoss(y_pred, y)
    loss.backward()
    optimizer.step()
    if i % 100 == 0:

```

```

print('Epoch {}, Loss = {}'.format(i, loss.data))

# Predict and plot
perdict = model(xx).data.numpy()
plt.scatter(x_d, y.detach().numpy(), color='r')
plt.plot(xx, perdict, 'k-')
plt.show()

```

From the result we can see the loss is decreasing. At the same time, the model can fit the data well.

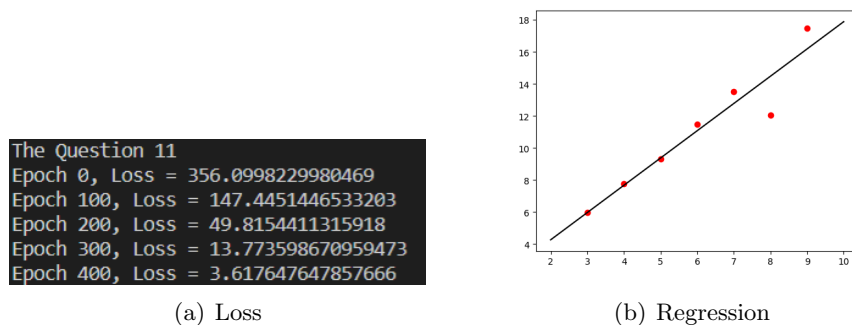


Figure 8: Result of Question 11

12 Implement logistic regression model and use autograd to optimize it by Pytorch.

```

def Q12():
    print('The Question 12')
    # We use a linear layer and a sigmoid layer to generate the regression model.
    class LogisticRegression(nn.Module):
        def __init__(self, input_size = 1, output_size = 1):
            super(LogisticRegression, self).__init__()
            self.linear = nn.Linear(input_size, output_size)
            self.sigmoid = nn.Sigmoid()

        def forward(self, x):
            return self.sigmoid(self.linear(x))

    iteration = 500
    # Set the data
    xx = np.linspace(1,4)

    x = np.linspace(2,8)
    y = [[0] * 25 + [1] * 25]
    xx = np.linspace(2,10)
    x_d = np.array(x)

    iteration = 500
    x = torch.FloatTensor(x)
    x = x.view(-1,1)
    x = Variable(x, requires_grad = True)

    xx = torch.FloatTensor(xx)

```

```

xx = xx.view(-1,1)
xx = Variable(xx)

y = torch.FloatTensor(y)
y = y.view(-1,1)
y = Variable(y)

Model = LogisticRegression()
BCELoss = nn.BCELoss()
optimizer = torch.optim.Adam(Model.parameters(), lr = 1e-2)

# Fit the data
for i in range(iteration):
    y_pred = Model(x)
    loss = BCELoss(y_pred, y)
    if i % 100 == 0:
        print('Epoch {}, Loss = {}'.format(i, loss.data))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

xx = torch.FloatTensor(xx)
xx = xx.view(-1,1)
xx = Variable(xx)
# Predict and plot
perdict = Model(xx).data.numpy()
xx = np.linspace(2,10)
plt.scatter(x_d, y.detach().numpy(), color='b')
plt.plot(xx, perdict, 'k-')
plt.show()

```

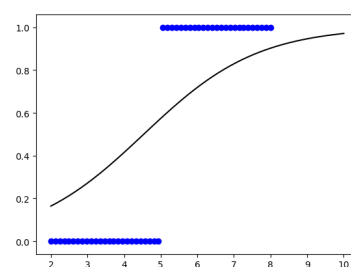
From the result we can see the loss is decreasing. At the same time, the model can also fit the data well.

```

The Question 12
Epoch 0, Loss = 1.2362279891967773
Epoch 100, Loss = 0.7301027178764343
Epoch 200, Loss = 0.5995914340019226
Epoch 300, Loss = 0.5010747313499451
Epoch 400, Loss = 0.4278576970100403

```

(a) Loss



(b) Regression

Figure 9: Result of Question 12

13 Implement linear SVM model for binary classification task and use autograd to optimize it by Pytorch.

```

def Q13_14():
    print('The Question 13/14')
    # Set the seed of random number

```

```

random.seed(0)
# Set up the SVM model
class SVM(nn.Module):
    def __init__(self, input_size = 2, output_size = 1):
        super(SVM, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    def forward(self, x):
        return self.linear(x)
iteration = 500
# Initialize data
x = np.r_[np.random.randn(20, 2) - [2,2], np.random.randn(20,2) + [2,2]]
y = [-1] * 20 + [1] * 20
x = torch.FloatTensor(x)
x = Variable(x, requires_grad = True)

y = torch.FloatTensor(y)
y = y.view(-1,1)
y = Variable(y)
Model = SVM()

optimizer = torch.optim.Adam(Model.parameters(), lr = 1e-2)
# Way1 to add the L2-Norm penalty
# optimizer = torch.optim.Adam(Model.parameters(), lr = 1e-2, weight_decay =
    0.01)
for i in range(iteration):
    y_pred = Model(x)

    L2_loss = 0
    # Way2 to add the L2-Norm penalty
    '''
    for param in Model.parameters():
        L2_loss += torch.sqrt(torch.sum(torch.pow(param, 2)))
    '''

    # Training
    loss = torch.sum(torch.clamp(1-y*y_pred, min=0)) + 0.01 * L2_loss
    if i % 100 == 0:
        print('Epoch {}, Loss = {}'.format(i, loss.data))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# Fetch the parameter in model to plot the figure
parameter = []
for para in Model.parameters():
    parameter.append(para.data.detach().numpy())

x_d = x.detach().numpy()
for i in range(20):
    plt.scatter(x_d[i][0], x_d[i][1], color='b')
for i in range(20,40):
    plt.scatter(x_d[i][0], x_d[i][1], color='r')
xx = np.linspace(-5,5)
yy = []
for xx_pro in xx:
    # Plot the SVM line

```

```

yy.append((0.5 - xx_pro * parameter[0][0][0] - parameter[1][0]) /
           parameter[0][0][1])
plt.plot(xx, yy, 'k-')
plt.show()

```

From the result we can see the loss is decreasing. At the same time, the model can also fit the data well. Note that the line are not the real separate line of SVM, it's only a line between 0-1 (we set 0.5 here) to indicate the separation.

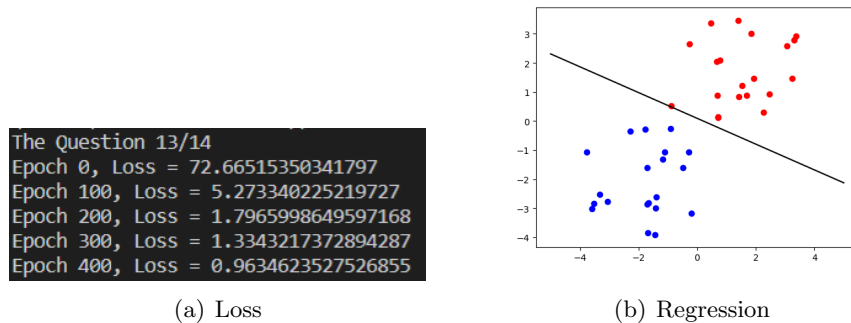


Figure 10: Result of Question 13

14 Add a Frobenius norm penalty for the weight w in your SVM model by two different ways.

There are two ways to add a Frobenius norm penalty for the weight w . One is add the parameters in the optimizer, and the other is to manually calculate the L2-norm of the model and add it to the loss function. You can refer to the code above.

15 Learn how to use linear regression, logistic regression, and SVM by scikit-learn.

After reading the official documents, I used scikit-learn to rebuild the model and plot it. The code and results are as follows

```

def Q15():
    print('The Question 15')
    # The linear regression model
    x = [3, 4, 5, 6, 7, 8, 9]
    x_d = np.array(x)
    x = np.array(x).reshape(-1,1)
    y = []
    xx_d = np.linspace(2,10)
    xx = np.linspace(2,10).reshape(-1,1)

    for i in x:
        y.append(i * 2 + random.normalvariate(0,1))
    reg = LinearRegression().fit(x, y)
    plt.scatter(x, y, color='b')
    yy = reg.predict(xx)
    plt.plot(xx, yy, 'k-')
    plt.show()

```

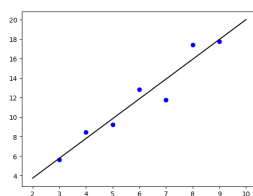
```

# The Logistic regression model
x_d = np.linspace(2,8)
y_d = [0 for i in range(25)] + [1 for i in range(25)]
x = np.linspace(2,8).reshape(-1,1)
y = np.array(y_d).reshape(-1,1)
xx = np.linspace(2,10)
x_d = np.array(x)
clf = LogisticRegression(random_state=0).fit(x,y)
yy = clf.predict_proba(x)[: ,1]
plt.scatter(x, y, color='b')
plt.plot(xx, yy, 'k-')
plt.show()
# The SVM model
random.seed(0)
x = np.r_[np.random.randn(20, 2) - [2,2], np.random.randn(20,2) + [2,2]] #
    Add Noise
y = np.array([-1] * 20 + [1] * 20)

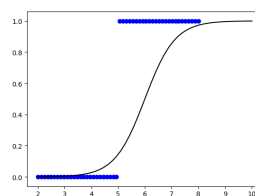
clf = SVC(kernel = 'linear')
clf.fit(x, y)
coef = clf.coef_[0]
a = - coef[0] / coef[1]
xx = np.linspace(-5, 5)

# plot the separation line
yy = a * xx - (clf.intercept_[0] / coef[1])
# plot the lower separation line
b = clf.support_vectors_[0]
y_d = a * xx + (b[1] - a * b[0])
# plot the upper separation line
b = clf.support_vectors_[-1]
y_u = a * xx + (b[1] - a * b[0])
plt.plot(xx, yy, 'k-')
plt.plot(xx, y_d, 'k--')
plt.plot(xx, y_u, 'k--')
# circle the points
plt.scatter(x[:, 0], x[:, 1], c = y, cmap = plt.cm.Paired)
plt.show()

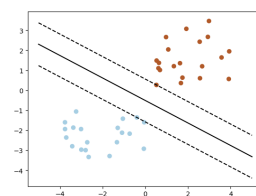
```



(a) Linear



(b) Logistic



(c) SVM

Figure 11: Result of Question 15

c

```
def Q17_20():
```

```

print('The Question 17-20')
# Create the class to load the dataset
class CIFAR10_Load(torch.utils.data.dataset.Dataset):
    def __init__(self, root, train = True):
        self.root = root
        self.train = train
        data = []
        labels = []
        if self.train:
            file_list = [self.root + '\\data_batch_' + str(i) for i in
                          range(1,6)]
        else:
            file_list = [self.root + '\\test_batch']
        for file in file_list:
            with open(file, 'rb') as fo:
                pack_dict = pickle.load(fo, encoding='bytes')
                data += [pack_dict[b'data']]
                labels += [pack_dict[b'labels']]
        # Transfer the list into numpy array
        data = np.concatenate(data)
        labels = np.concatenate(labels)
        self.data = np.transpose(np.reshape(data, [-1, 3, 32, 32]), [0, 2, 3,
                                                                    1])
        # Transfer the data into (batch_size, 32, 32, 3) so that we can
        # visualize it
        self.labels = labels

    def __len__(self):
        return self.data.shape[0]
    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]

# Load the data
train = CIFAR10_Load(root = './cifar-10-python\\cifar-10-batches-py', train=
    True)
test = CIFAR10_Load(root = './cifar-10-python\\cifar-10-batches-py', train=
    False)
TrainLoader = torch.utils.data.DataLoader(train, batch_size=5000,
    num_workers=0, pin_memory=False)

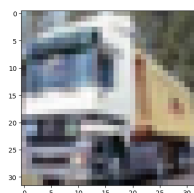
print('Start Processing')
time1 = time.time()
total = []
for X_batch, y_batch in TrainLoader:
    # Question 16
    '''
    plt.imshow(X_batch[1])
    plt.show()
    '''

    # Question 20
    Mean = np.mean(np.reshape(X_batch.numpy(), [-1,3]), axis = 0).tolist()
    Std = np.std(np.reshape(X_batch.numpy(), [-1,3]), axis = 0).tolist()
    print('Mean of CIFAR-10 training set within each RGB channel is: ', Mean)
    print('Std of CIFAR-10 training set within each RGB channel is: ', Std)

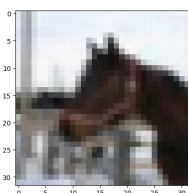
# Question 19
time2 = time.time() - time1
print(time2)

```

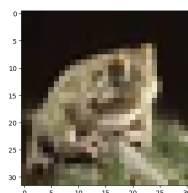
After downloading the dataset, we can visualize some of the pictures.



(a) Picture 1



(b) Picture 2



(c) Picture 3

Figure 12: Some picture on CIFAR-10 dataset

17 Write a dataset class for loading CIFAR-10. Make sure it could be transferred to Pytorch Dataloader.

Also, in the code above, we write a dataset class for loading CIFAR-10 to let it available for torch.

18 Read and learn how to use torchvision.transforms to transform images.

```
def Q18():
    img = Image.open('./picture.jpg')

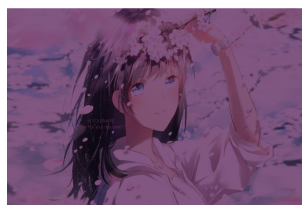
    transform=transforms.Compose([
        transforms.Scale([1024,1550]),
        transforms.ToTensor(),
        transforms.Normalize(mean=[2,3,2], std=[2, 3, 2])])

    img2 = transform(img)
    img_2 = transforms.ToPILImage()(img2).convert('RGB')
    img_2.show()
```

Here are the pictures before and after the modification. It can be seen that we can change pictures by changing the mean and variance of the pixel values are cropped and modified.



(a) Before transformation



(b) After transformation

Figure 13: Pictures before and after transformation

19 Run one epoch for loading CIFAR-10 with Pytorch Dataloader and test the loading time of different batchsize (1, 4, 64, 1024), different numworkers (0,1,4,16), and whether use pin_memory or not.

The result is as followed, note that when numworker ≥ 1 , the program can only run on Linux system. When the batchsize increasing, the loading time decreasing, and open the pin_memory also can reduce the loading time. However, the Numworker should match the number of CPU cores.

Table 1: Loading Time(Numworker = 4)

Batchsize	1	4	64	1024
Time(s)	33.82	10.83	3.99	3.18

Table 2: Loading Time(Batchsize = 64)

Numworker	0	1	4	16
Time(s)	4.42	4.24	3.99	4.06

Table 3: Loading Time(Batchsize = 64, Numworker = 4)

Pinmemory	True	False
Time(s)	4.13	3.65

20 Calculate the mean and std of CIFAR-10' training set within each RGB channel.

From the code above, we can calculate the mean and std of CIFAR-10' training set within each RGB channel. The result is as followed.

21 Image to character painting

```
def Q21():
    def process_bar(percent, start_str='', end_str='', total_length=0):
        # print the process bar
        bar = ''.join(["\033[31m%s\033[0m"%'-' * int(percent * total_length)] +
            '>')
        bar = '\r' + start_str + bar.ljust(total_length) + '
{:0>4.1f}%|'.format(percent*100) + end_str
        print(bar, end='', flush=True)
    print('Question 21 is processing.')
    def transfer(r,g,b):
        blank = 50
        gap = (255 - blank) / 26
        # the piecewise linear transformation function
        j = 0.2126 * r + 0.7152 * g + 0.0722 * b
        character = ''
        if j < blank:
            character = ' '
        else:
```

```

The Question 17-20
Start Processing
Mean of CIFAR-10' training set within each RGB channel is: [125.2715861328125, 122.7609314453125
, 113.3391826171875]
Std of CIFAR-10' training set within each RGB channel is: [63.13781215090041, 62.34681560083387,
67.00009668747751]

```

Figure 14: Result of Question 20

```

        character = chr(65 + int((j-blank)/gap))
        if character == 'Z':
            character = ' '
        return character
img = Image.open("./pic.jpg")
pix = img.load()
width = img.size[0]
height = img.size[1]
new_pic = Image.new("RGB", (width, height))
new_arr = [['' for i in range(width)] for j in range(height)]

for x in range(width):
    for y in range(height):
        r, g, b = pix[x, y][0:3]
        n = transfer(r,g,b)
        # transfer the pixel in different cernel into a new one
        new_arr[y][x] = n
        #f.write(n)
    #f.write('\n')
    time.sleep(0.1)
    end_str = '100%'
    process_bar(x/width, start_str='', end_str=end_str, total_length=15)
f = open("./Test.txt", 'w')
for i in range(height):
    for j in range(width):
        f.write(new_arr[i][j])
    f.write('\n')
f.close()

```

This is the result. We can find that when using characters to replace the corresponding gray value, only some rough outlines of the picture are retained.

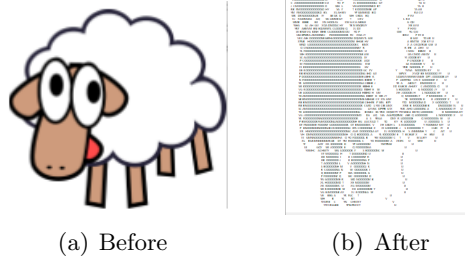


Figure 15: Pictures before and after transformation

22 Numpy exercises.

```
def Q22():

    # item 1
    origin = np.random.normal(0,1,(10,2))
    r = []
    Angle = []
    for i in range(10):
        coor = [0, 0]
        coor[0] = origin[i,0]
        coor[1] = origin[i,1]
        r = np.sqrt(coor[0]*coor[0] + coor[1]*coor[1])
        Angle = np.arctan(coor[1]/coor[0]) * 180 / math.pi
        # Dealing with different quadrants
        if coor[0] > 0 and coor[1] > 0:
            print(origin[i,:], '-->', (r, Angle))
        elif coor[0] < 0 and coor[1] > 0:
            print(origin[i,:], '-->', (r, Angle+180))
        elif coor[0] < 0 and coor[1] < 0:
            print(origin[i,:], '-->', (r, Angle+180))
        else:
            print(origin[i,:], '-->', (r, 360 + Angle))

    # item 2
    class Symmetric(np.ndarray):
        def __setitem__(self, index, value):
            i = index[0]
            j = index[1]
            # Rewrite the setitem function
            super(Symmetric, self).__setitem__((i,j), value)
            super(Symmetric, self).__setitem__((j,i), value)
    def generate(shape):
        # generate a symmetric
        result = np.zeros(shape).view(Symmetric)
        for i in range(shape[0]):
            for j in range(i, shape[0]):
```

```

        result[i,j] = random.randint(1,10)
    return result

a = generate((5,5))
print(a)
# Test whether we can change the value symmetrically
a[1,0] = 10
print(a)

# item 3
def distance_point(a,b):
    return np.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)
def distance(p0,p1,p2):
    # Heron's formula
    l1 = distance_point(p0,p1)
    l2 = distance_point(p1,p2)
    l3 = distance_point(p2,p0)
    p = (l1 + l2 + l3) / 2
    s = np.sqrt(p * (p - l1) * (p - l2) * (p - l3))
    return 2 * s / l1

P0 = np.random.uniform(-5, 5, (5,2))
P1 = np.random.uniform(-5, 5, (5,2))
p = np.random.uniform(-5, 5, (5,2))
for i in range(5):
    dist = distance(P0[i,],P1[i,],p[i,])
    print(p[i,], '--> (', P0[i,], P1[i,], ') = ', dist)

```

- Here we write a new class which inherit the numpy.ndarray, and using the "super" item to replace the "__setitem__" to change the way numpy load the data to a symmetric way.
- Here we use the Heron's formula to calculate the area of triangle, so that we can find the distance.

Here are the results of question 22.

```

[ 0.40155823 -0.70348324] --> (0.8100232601222233, 299.71833898143643)
[-0.84253767 -0.57057522] --> (1.0175587492482596, 214.1062632989004)
[ 0.38181959 -2.14372163] --> (2.1774592138179534, 280.0990830696629)
[-0.01357886 -0.81000504] --> (0.8101188534537417, 269.0395877590455)
[-0.0408538 -1.82023255] --> (1.820690960928453, 268.7142535060734)
[ 0.55749228 -0.28101481] --> (0.6243131931059316, 333.24872443795357)
[-0.62605158 -2.33423919] --> (2.416736062099818, 254.98639563043594)
[ 2.03061132 -3.72296356] --> (0.2433930913876824, 236.87172542324979)
[-1.99819895  2.05139374] --> (2.86374148985928, 134.24741511688785)
[2.41440525  0.61079583] --> (2.490466678510281, 14.19681910552192)

```

(a) Q22_1

```

[[ 4.  9.  6.  6.  8.]
 [ 9.  6.  5.  7.  6.]
 [ 6.  5.  6. 10.  2.]
 [ 6.  7. 10.  4.  9.]
 [ 8.  6.  2.  9. 10.]]
[[ 4. 10.  6.  6.  8.]
 [10.  6.  5.  7.  6.]
 [ 6.  5.  6. 10.  2.]
 [ 6.  7. 10.  4.  9.]
 [ 8.  6.  2.  9. 10.]]

```

(b) Q22_2

```

[ 0.55893353 -0.66417571] --> ( [ 2.8188131 -1.04676867] [-2.28648359 -2.08869738] ) = 0.8267643180134544
[-1.82818798 -2.2988158 ] --> ( [-1.95683296  0.59275912] [-2.75039016  1.57530954] ) = 1.7167464326891773
[ 2.03061132 -3.72296356] --> ( [-4.48200293 -0.83008518] [4.62419243  3.1626598 ] ) = 5.264599345823165
[-0.93311558 -3.25228307] --> ( [ 0.39497191 -4.77771063] [-0.11801734  0.66820007] ) = 1.1791766431958586
[-4.85478801 -4.91902917] --> ( [1.61196678  2.75173481] [ 1.7088608 -3.78227431] ) = 6.57978253794371

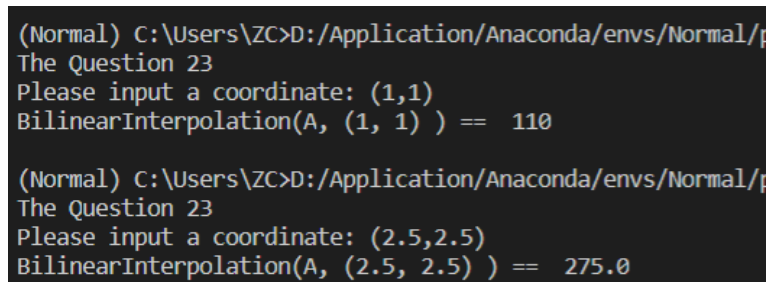
```

(c) Q22_3

Figure 16: Pictures before and after transformation

23 Bi-linear Interpolation

```
def Q23():
    print('The Question 23')
    A = [[110, 120, 130], [210, 220, 230], [310, 320, 330]]
    coord = eval(input('Please input a coordinate: '))
    d1 = coord[0] - math.floor(coord[0])
    d2 = coord[1] - math.floor(coord[1])
    f1 = math.floor(coord[0])
    f2 = math.floor(coord[1])
    result = A[f1-1][f2-1] * (1-d1) * (1-d2) + \
        A[f1][f2-1] * d1 * (1-d2) + \
        A[f1-1][f2] * (1-d1) * d2 + \
        A[f1][f2] * d1 * d2
    print('BilinearInterpolation(A, ', coord, ') == ', result)
```



```
(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/p
The Question 23
Please input a coordinate: (1,1)
BilinearInterpolation(A, (1, 1) ) == 110

(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/p
The Question 23
Please input a coordinate: (2.5,2.5)
BilinearInterpolation(A, (2.5, 2.5) ) == 275.0
```

Figure 17: Result of Question 23

24 Cartesian product.

```
def Q24():
    print('The Question 24')
    def descartes(v,n):
        # Calculate the Cartesian product between 2 lists
        return [v[i]+[n[j]] for i in range(len(v)) for j in range(len(n)) ]
    # Input the data
    l = input('Please input the vectors: ')
    l = l.lstrip('[').rstrip(']').split(', ')
    origin = []
    for s in l:
        s = s.split(' ')
        ints = []
        for ch in s:
            ints.append(int(ch))
        origin.append(ints)
    result = [[i] for i in origin[0]]
    for i in range(1, len(origin)):
        result = descartes(result, origin[i])
    print(result)
```

Here we only need to calculate the Cartesian product between 2 lists and then using the result to calculate with another one until the end. The results is as followed.

```
(Normal) C:\Users\ZC>D:/Application/Anaconda/envs/Normal/python.exe d:/Documents/课程介绍/神经网络与深度学习/Project1/Project1.py
The Question 24
Please input the vectors: [1, 2, 3], [4, 5], [6, 7]
[[1, 4, 6], [1, 4, 7], [1, 5, 6], [1, 5, 7], [2, 4, 6], [2, 4, 7], [2, 5, 6], [2, 5, 7], [3, 4, 6], [3, 4, 7], [3, 5, 6], [3, 5, 7]]
```

Figure 18: Result of Question 24

25 Extracting a subpart of an array

```
def Q25():
    print('The Question 25')
    def subpart(Z, shape, fill, position):
        b_coord = (position[0] - math.ceil(shape[0]/2), position[1] -
                    math.ceil(shape[1]/2))
        # Navigate to start position
        matrix = []
        for i in range(shape[0]):
            row = []
            for j in range(shape[1]):
                write_i = i + b_coord[0]
                write_j = j + b_coord[1]
                # Padding
                if (write_i < 0) or (write_j < 0) or (write_i >= len(Z)) or
                    (write_j >= len(Z[i])):
                    row.append(0)
                else:
                    row.append(Z[write_i][write_j])
            matrix.append(row)
        return matrix

    print('The Question 25')
    Z = np.random.randint(0, 10, (5, 5))
    print(Z)
    shape = (4, 4)
    fill = 0
    position = (1, 1)
    result = subpart(Z, shape, fill, position)
    for i in range(len(result)):
        print(result[i])
```

In this question, we first located to where the matrix begin, and then decide whether we need to use the padding based on the location.

```
The Question 25
[[9 9 9 4 5]
 [7 1 3 6 8]
 [0 8 7 8 7]
 [0 2 1 6 9]
 [7 9 0 0 8]]
[0, 0, 0, 0]
[0, 9, 9, 9]
[0, 7, 1, 3]
[0, 0, 8, 7]
```

Figure 19: Result of Question 25

26. Matrix operations

```
def Q26():
    print('The Question 26')
    def add(A,B):
        row_len = len(A)
        column_len = len(B[0])
        cross_len = len(B)
        result = [[0 for j in range(column_len)] for i in range(row_len)]
        for i in range(row_len):
            for j in range(column_len):
                result[i][j] = A[i][j] + B[i][j]
        return result

    def subtract(A,B):
        row_len = len(A)
        column_len = len(B[0])
        cross_len = len(B)
        result = [[0 for j in range(column_len)] for i in range(row_len)]
        for i in range(row_len):
            for j in range(column_len):
                result[i][j] = A[i][j] - B[i][j]
        return result

    def scalar_multiply(A,B):
        row_len = len(A)
        column_len = len(A[0])
        result = [[0 for j in range(column_len)] for i in range(row_len)]
        for i in range(row_len):
            for j in range(column_len):
                result[i][j] = A[i][j] * B
        return result

    def multiply(A,B):
        row_len = len(A)
        column_len = len(B[0])
        cross_len = len(B)
        result = [[0 for j in range(column_len)] for i in range(row_len)]
        for i in range(row_len):
            for j in range(column_len):
                for k in range(cross_len):
                    temp = A[i][k] * B[k][j]
                    result[i][j] += temp
        return result

    def identity(n):
        result = [[0 for j in range(n)] for i in range(n)]
        for i in range(n):
            result[i][i] = 1
        return result

    def transpose(A):
        row_len = len(A)
        column_len = len(A[0])
        result = [[[0] for j in range(row_len)] for i in range(column_len)]
```

```

    for i in range(column_len):
        for j in range(row_len):
            result[i][j] = A[j][i]
    return result

def inverse(A):
    def det(mat):
        # Calculate the determinant of the matrix
        if len(mat) <= 0:
            return None
        if len(mat) == 1:
            return mat[0][0]
        else:
            total = 0
            for i in range(len(mat)):
                n = [[row[r] for r in range(len(mat)) if r != i] for row in
                    mat[1:]]
                if i % 2 == 0:
                    total += mat[0][i] * det(n)
                else:
                    total -= mat[0][i] * det(n)
            return total
    row_len = len(A)
    column_len = len(A[0])
    det_all = det(A)
    result = [[[] for j in range(row_len)] for i in range(column_len)]
    for i in range(row_len):
        for j in range(column_len):
            # Generate the Adjoint matrix
            cofactor = [[A[b][a] for a in range(column_len) if a != j] for b
                in range(row_len) if b != i]
            result[j][i] = (-1) ** (i+j) * det(cofactor) / det_all
    return result

matrix_a = [[12, 10], [3, 9]]
matrix_b = [[3, 4], [7, 4]]
matrix_c = [[11, 12, 13, 14], [21, 22, 23, 24], [31, 32, 33, 34], [41, 42,
    43, 44]]
matrix_d = [[3, 0, 2], [2, 0, -2], [0, 1, 1]]
print('add(matrix_a, matrix_b) == ', add(matrix_a, matrix_b))
print('subtract(matrix_a, matrix_b) == ', subtract(matrix_a, matrix_b))
print('scalar_multiply(matrix_b, 3) == ', scalar_multiply(matrix_b, 3))
print('multiply(matrix_a, matrix_b) == ', multiply(matrix_a, matrix_b))
print('identity(3) == ', identity(3))
print('transpose(matrix_c) == ', transpose(matrix_c))
print('inverse(matrix_d) == ', inverse(matrix_d))

```

The addition, subtraction, multiplication, transposition of the matrix, and construction of the identity matrix can be written by what they defined. However, when we calculate the inverse of matrix, we need to use the formula $A^{-1} = \frac{A^*}{|A|}$, where A^* and $|A|$ denote the ad-joint matrix and determinant of A. Here is the result.

27 Greatest common divisor

```

The Question 26
add(matrix_a, matrix_b) == [[15, 14], [10, 13]]
subtract(matrix_a, matrix_b) == [[9, 6], [-4, 5]]
scalar_multiply(matrix_b, 3) == [[9, 12], [21, 12]]
multiply(matrix_a, matrix_b) == [[106, 88], [72, 48]]
identity(3) == [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
transpose(matrix_c) == [[11, 21, 31, 41], [12, 22, 32, 42], [13, 23, 33, 43], [14, 24, 34, 44]]
inverse(matrix_d) == [[0.2, 0.2, 0.0], [-0.2, 0.3, 1.0], [0.2, -0.3, 0.0]]

```

Figure 20: Result of Question 26

```

def Q27(a,b):
    print('The Question 27')
    p = max(abs(a),abs(b))
    q = min(abs(a),abs(b))
    if q == 0:
        result = p
        print('GCD(' ,a, ',' ,b, ') =' ,p)
        return
    temp = p % q
    while (temp != 0):
        p = q
        q = temp
        temp = p % q
    print('GCD(' ,a, ',' ,b, ') =' ,q)
    return

```

We first find the absolute value of the two numbers and compare it, and then use the division algorithm to find the greatest common divisor. Here is the result.

```

The Question 27
GCD( 3 , 5 ) = 1
The Question 27
GCD( 6 , 3 ) = 3
The Question 27
GCD( -2 , 6 ) = 2
The Question 27
GCD( 0 , 3 ) = 3

```

Figure 21: Result of Question 27

28 Find all consecutive positive number sequences whose sum is N

```

def Q28(N):
    print('The Question 28')
    i = 0
    while i < int(math.sqrt(2*N)):
        i += 1
        if (2 * N / i + 1 - i) % 2 == 0:
            start = int((2 * N / i + 1 - i) / 2)
            print(list(range(start,start+i)))
    return

```

Here we use the summation formula of the sequence of equal difference numbers. So that we can calculate the first number of the sequence.

```
The Question 28
[1000]
[198, 199, 200, 201, 202]
[55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
[28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52]
```

Figure 22: Result of Question 28

29 Password checking

```
def Q29():
    print('The Question 29')
    char_list = input('Please input: ')
    char_list = char_list.split(',')
    legal = []
    for s in char_list:
        if 6 <= len(s) <= 12:
            char = re.findall(r'[a-z]', s)
            upperchar = re.findall(r'[A-Z]', s)
            num = re.findall(r'[0-9]', s)
            spe = re.findall(r'[\$#@]', s)
            if char and upperchar and num and spe:
                legal.append(s)
    string = ','.join(legal)
    print(string)
```

```
The Question 29
Please input: ABd1234@1,a F1#,2w3E*,2We3345
ABd1234@1
```

Figure 23: Result of Question 29