

assignment_06

October 14, 2021

1 Multi-class classification based on Softmax and Cross-Entropy using pytorch

1.1 import libraries

```
[ ]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torch.utils.data import Dataset
from torchvision import datasets, transforms
import torchvision.transforms.functional as F
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import tqdm
import random
import os
```

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

1.2 load data

```
[ ]: directory_data = 'drive/MyDrive/'
filename_data = 'assignment_06_data.npz'
data = np.load(os.path.join(directory_data, filename_data))

x_train = data['x_train']
y_train = data['y_train']

x_test = data['x_test']
y_test = data['y_test']

num_data_train = x_train.shape[0]
num_data_test = x_test.shape[0]
```

```

print('*****')
print('size of x_train :', x_train.shape)
print('size of y_train :', y_train.shape)
print('*****')
print('size of x_test :', x_test.shape)
print('size of y_test :', y_test.shape)
print('*****')
print('number of training image :', x_train.shape[0])
print('height of training image :', x_train.shape[1])
print('width of training image :', x_train.shape[2])
print('*****')
print('number of testing image :', x_test.shape[0])
print('height of testing image :', x_test.shape[1])
print('width of testing image :', x_test.shape[2])
print('*****')

```

```

*****
size of x_train : (20000, 32, 32)
size of y_train : (20000,)
*****
size of x_test : (8000, 32, 32)
size of y_test : (8000,)
*****
number of training image : 20000
height of training image : 32
width of training image : 32
*****
number of testing image : 8000
height of testing image : 32
width of testing image : 32
*****

```

1.3 number of classes

```

[ ]: print('*****')
      print('number of classes :', len(set(y_train)))
      print('*****')

```

```

*****
number of classes : 10
*****

```

1.4 hyper-parameters

```
[ ]: device          = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

number_epoch      = 100
size_minibatch    = 20
learning_rate     = 0.01
weight_decay      = 1e-3
```

1.5 custom data loader for the PyTorch framework

```
[ ]: class dataset(Dataset):

    def __init__(self, image, label):

        self.image = image
        self.label = label.astype(np.long)

    def __getitem__(self, index):

        image = self.image[index, :, :]
        label = self.label[index, ]

        image = torch.FloatTensor(image).unsqueeze(dim=0)
        label = torch.LongTensor([label])

        return image, label

    def __len__(self):

        return self.image.shape[0]

    def collate_fn(self, batch):

        images = list()
        labels = list()

        for b in batch:
            images.append(b[0])
            labels.append(b[1])

        images = torch.stack(images, dim=0)
        labels = torch.stack(labels, dim=0).squeeze()

        return images, labels
```

1.6 construct datasets and dataloaders for training and testing

```
[ ]: dataset_train = dataset(x_train, y_train)
dataset_test = dataset(x_test, y_test)

dataloader_train = torch.utils.data.DataLoader(dataset_train,
    ↳batch_size=size_minibatch, shuffle=True, drop_last=True,
    ↳collate_fn=dataset_train.collate_fn)
dataloader_test = torch.utils.data.DataLoader(dataset_test,
    ↳batch_size=size_minibatch, shuffle=True, drop_last=True,
    ↳collate_fn=dataset_test.collate_fn)
```

1.7 shape of the data when using the data loader

```
[ ]: image, label = next(iter(dataloader_train))
print('*****')
print('size of mini-batch of the image:', image.shape)
print('*****')
print('size of mini-batch of the label:', label.shape)
print('*****')
```

```
*****
size of mini-batch of the image: torch.Size([20, 1, 32, 32])
*****
size of mini-batch of the label: torch.Size([20])
*****
```

1.8 class for the neural network

```
[ ]: class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()

        self.feature = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
        )
```

```

self.classifier = nn.Sequential(
    nn.Linear(64*8*8,128),
    nn.BatchNorm1d(128),
    nn.ReLU(),
    nn.Linear(128,64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Linear(64,32),
    nn.BatchNorm1d(32),
    nn.ReLU(),
    nn.Linear(32,10),
)

self.network = nn.Sequential(
    self.feature,
    nn.Flatten(),
    self.classifier,
)

self.initialize()

def initialize(self):

    for m in self.network.modules():

        if isinstance(m, nn.Conv2d):

            nn.init.xavier_uniform_(m.weight)
            nn.init.constant_(m.bias, 1)

        elif isinstance(m, nn.Linear):

            nn.init.xavier_uniform_(m.weight)
            nn.init.constant_(m.bias, 1)

def forward(self, input):

    output = self.network(input)

    return output

```

1.9 build network

```
[ ]: classifier = Classifier().to(device)
optimizer = torch.optim.SGD(classifier.parameters(), lr=learning_rate,
    ↪weight_decay=weight_decay)
```

1.10 print the defined neural network

```
[ ]: print(classifier)
```

```
Classifier(
  (feature): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=4096, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=64, out_features=32, bias=True)
    (7): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (8): ReLU()
    (9): Linear(in_features=32, out_features=10, bias=True)
  )
  (network): Sequential(
    (0): Sequential(
      (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
```

```

track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(1): Flatten(start_dim=1, end_dim=-1)
(2): Sequential(
  (0): Linear(in_features=4096, out_features=128, bias=True)
  (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU()
  (3): Linear(in_features=128, out_features=64, bias=True)
  (4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (5): ReLU()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (8): ReLU()
  (9): Linear(in_features=32, out_features=10, bias=True)
)
)
)

```

1.11 compute the prediction

```

[ ]: def compute_prediction(model, input):

    prediction = model(input)

    return prediction

```

1.12 compute the loss

```
[ ]: def compute_loss(prediction, label):  
  
    criterion = nn.CrossEntropyLoss()  
    loss      = criterion(prediction, label)  
    loss_value = loss.item()  
  
    return loss, loss_value
```

1.13 compute the accuracy

```
[ ]: def compute_accuracy(prediction, label):  
    #  
    =====  
    #  
    # complete the function body  
  
    argmax_value = torch.argmax(prediction, dim=1)  
    compare      = torch.eq(argmax_value, label).tolist()  
    accuracy     = sum(compare)/len(label)  
    #  
    =====  
    #  
  
    return accuracy
```

1.14 variables for the learning curve

```
[ ]: loss_mean_train    = np.zeros(number_epoch)  
loss_std_train        = np.zeros(number_epoch)  
accuracy_mean_train   = np.zeros(number_epoch)  
accuracy_std_train    = np.zeros(number_epoch)  
  
loss_mean_test        = np.zeros(number_epoch)  
loss_std_test         = np.zeros(number_epoch)  
accuracy_mean_test     = np.zeros(number_epoch)  
accuracy_std_test      = np.zeros(number_epoch)
```

1.15 train and test

```
[ ]: #  
    =====  
    #  
    # iterations for epochs  
    #
```



```

#
→ =====
for i in tqdm(range(number_epoch)):

    #
    → =====
    #
    # training
    #
    #
    → =====
    loss_train_epoch      = []
    accuracy_train_epoch  = []

    classifier.train()

    for index_batch, (image_train, label_train) in enumerate(dataloader_train):

        image_train = image_train.to(device)
        label_train = label_train.to(device)

        prediction_train      = compute_prediction(classifier,
→ image_train)
        loss_train, loss_value_train = compute_loss(prediction_train,
→ label_train)
        accuracy_train        = compute_accuracy(prediction_train,
→ label_train)

        optimizer.zero_grad()
        loss_train.backward()
        optimizer.step()

        loss_train_epoch.append(loss_value_train)
        accuracy_train_epoch.append(accuracy_train)

    loss_mean_train[i]      = np.mean(loss_train_epoch)
    loss_std_train[i]       = np.std(loss_train_epoch)

    accuracy_mean_train[i]  = np.mean(accuracy_train_epoch)
    accuracy_std_train[i]   = np.std(accuracy_train_epoch)

    #
    → =====
    #
    # testing
    #

```

```

#
→ =====
loss_test_epoch      = []
accuracy_test_epoch  = []

classifier.eval()

for index_batch, (image_test, label_test) in enumerate(dataloader_test):

    image_test = image_test.to(device)
    label_test = label_test.to(device)

    prediction_test      = compute_prediction(classifier, image_test)
    loss_test, loss_value_test = compute_loss(prediction_test, label_test)
    accuracy_test        = compute_accuracy(prediction_test,
→label_test)

    loss_test_epoch.append(loss_value_test)
    accuracy_test_epoch.append(accuracy_test)

loss_mean_test[i]      = np.mean(loss_test_epoch)
loss_std_test[i]       = np.std(loss_test_epoch)

accuracy_mean_test[i]   = np.mean(accuracy_test_epoch)
accuracy_std_test[i]    = np.std(accuracy_test_epoch)

```

```

0%|          | 0/100 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-
packages/torch/nn/functional.py:718: UserWarning: Named tensors and all their
associated APIs are an experimental feature and subject to change. Please do not
use them for anything important until they are released as stable. (Triggered
internally at /pytorch/c10/core/TensorImpl.h:1156.)
    return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)
100%|         | 100/100 [07:07<00:00, 4.28s/it]

```

1.16 plot curve

```

[ ]: def plot_curve_error(data_mean, data_std, x_label, y_label, title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    alpha = 0.3

    plt.plot(range(len(data_mean)), data_mean, '-', color = 'red')
    plt.fill_between(range(len(data_mean)), data_mean - data_std, data_mean +
→data_std, facecolor = 'blue', alpha = alpha)

```

```
plt.xlabel(x_label)
plt.ylabel(y_label)

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

```
[ ]: def print_curve(data, index):

    for i in range(len(index)):

        idx = index[i]
        val = data[idx]

        print('index = %2d, value = %12.10f' % (idx, val))
```

```
[ ]: def get_data_last(data, index_start):

    data_last = data[index_start:]

    return data_last
```

```
[ ]: def get_max_last_range(data, index_start):

    data_range = get_data_last(data, index_start)
    value = data_range.max()

    return value
```

```
[ ]: def get_min_last_range(data, index_start):

    data_range = get_data_last(data, index_start)
    value = data_range.min()

    return value
```

2 functions for presenting the results

```
[ ]: def function_result_01():

    plot_curve_error(loss_mean_train, loss_std_train, 'epoch', 'loss', 'loss_↓
↪(training)')
```

```
[ ]: def function_result_02():

    plot_curve_error(accuracy_mean_train, accuracy_std_train, 'epoch',
↳ 'accuracy', 'accuracy (training)')

[ ]: def function_result_03():

    plot_curve_error(loss_mean_test, loss_std_test, 'epoch', 'loss', 'loss
↳ (testing)')

[ ]: def function_result_04():

    plot_curve_error(accuracy_mean_test, accuracy_std_test, 'epoch',
↳ 'accuracy', 'accuracy (testing)')

[ ]: def function_result_05():

    data_last = get_data_last(loss_mean_train, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)

[ ]: def function_result_06():

    data_last = get_data_last(accuracy_mean_train, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)

[ ]: def function_result_07():

    data_last = get_data_last(loss_mean_test, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)

[ ]: def function_result_08():

    data_last = get_data_last(accuracy_mean_test, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)

[ ]: def function_result_09():

    value = get_max_last_range(accuracy_mean_train, -10)
    print('best training accuracy = %12.10f' % (value))

[ ]: def function_result_10():

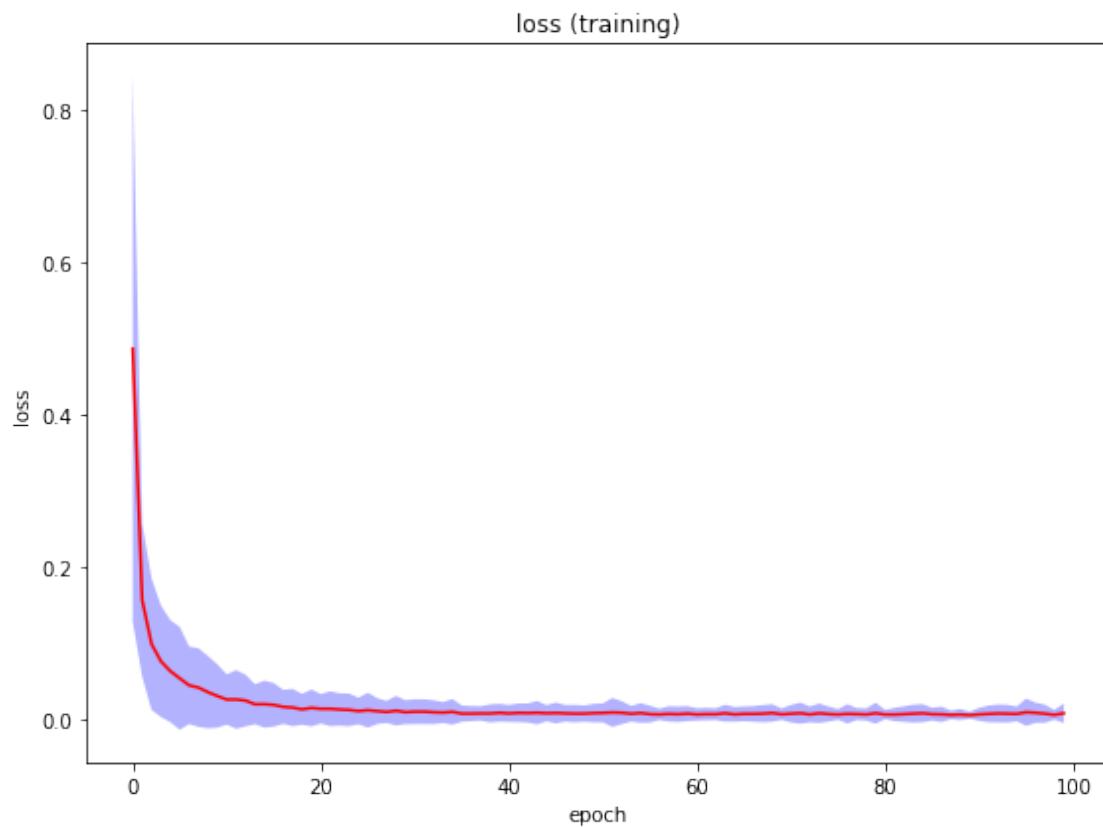
    value = get_max_last_range(accuracy_mean_test, -10)
```

```
print('best testing accuracy = %12.10f' % (value))
```

3 RESULTS

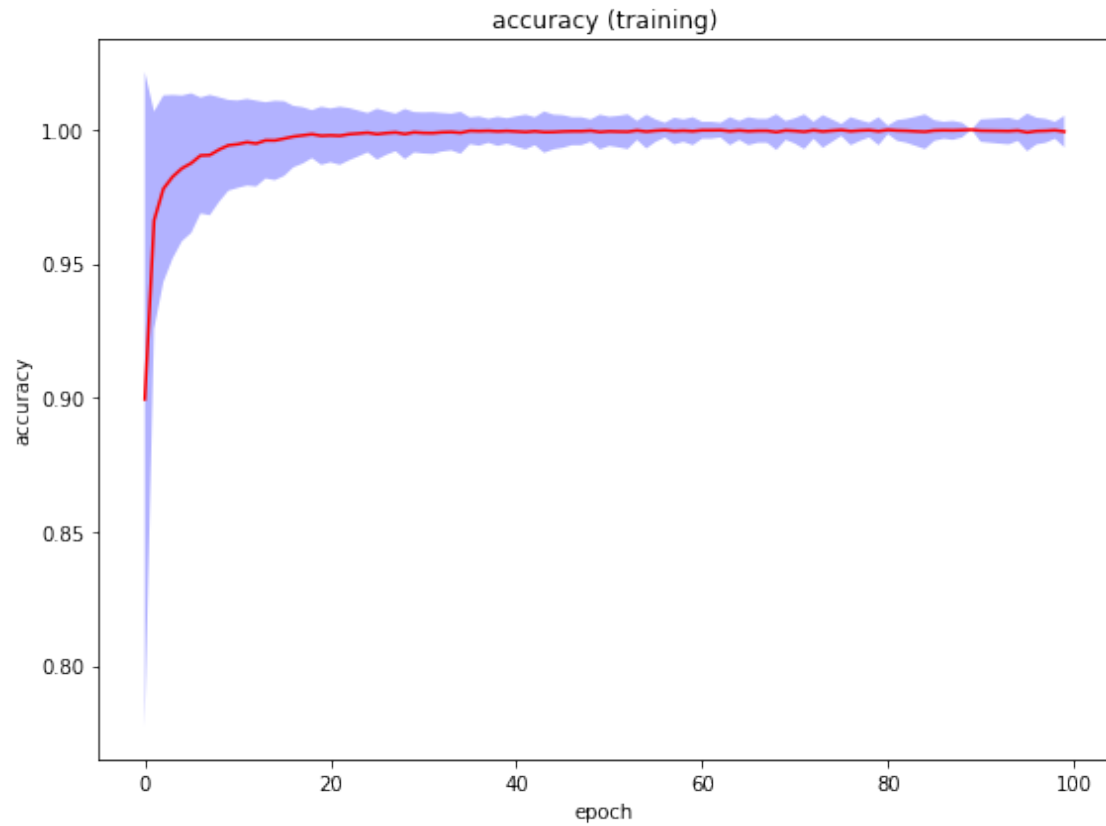
3.1 # 01. plot the training loss curve (mean, std)

```
[ ]: function_result_01()
```



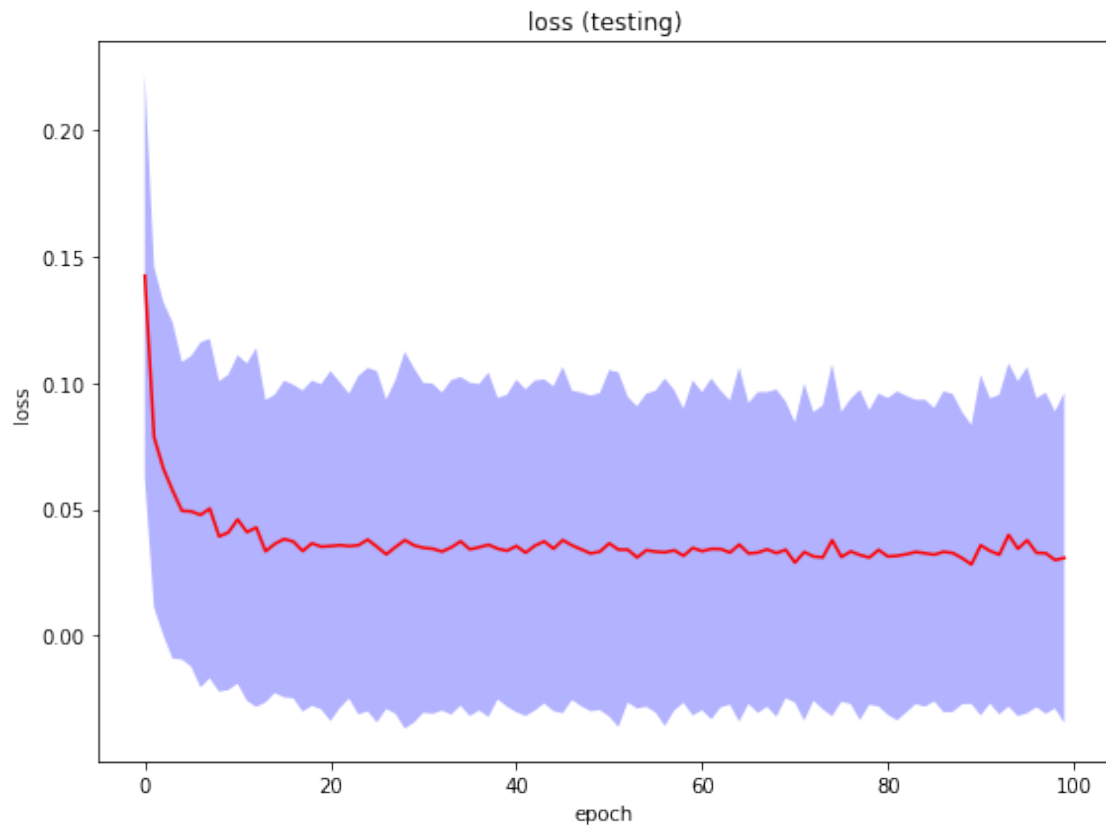
3.2 # 02. plot the training accuracy curve (mean, std)

```
[ ]: function_result_02()
```



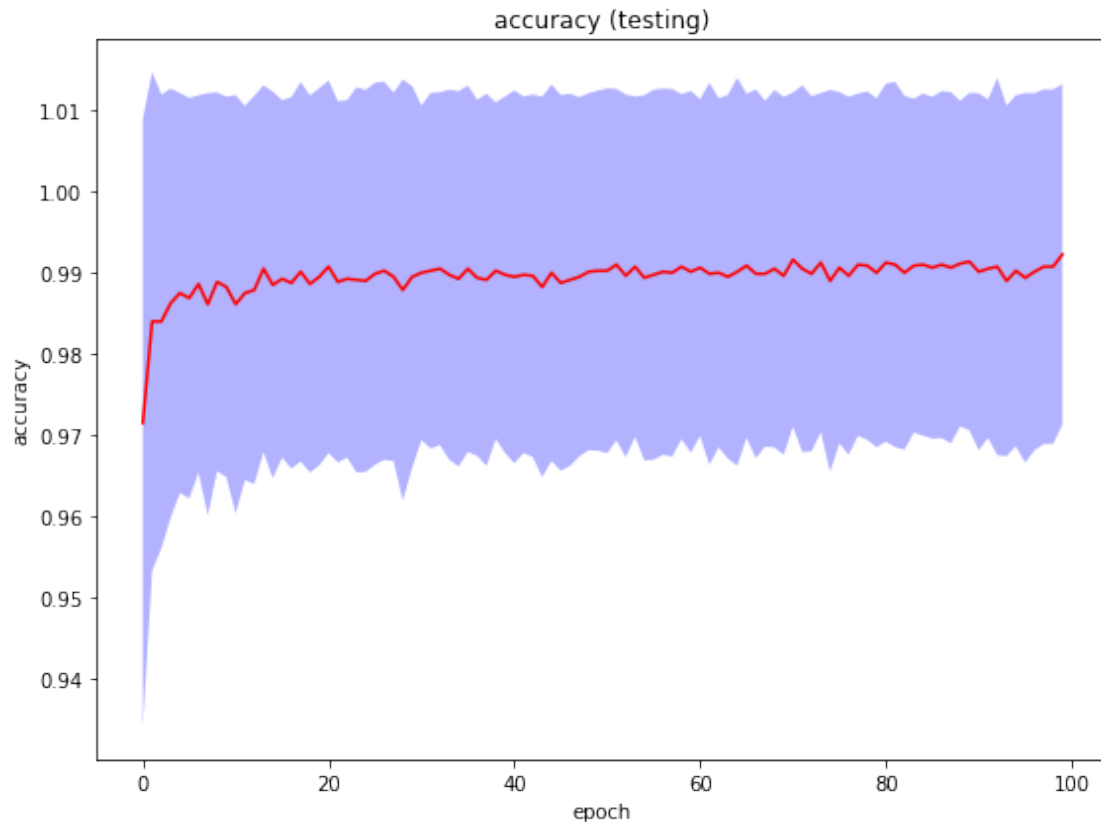
3.3 # 03. plot the testing loss curve (mean, std)

```
[ ]: function_result_03()
```



3.4 # 04. plot the testing accuracy curve (mean, std)

```
[ ]: function_result_04()
```



3.5 # 05. print the training (mean) loss over batches at last 10 epochs

```
[ ]: function_result_05()
```

```
index = 0, value = 0.0061364475
index = 1, value = 0.0067260348
index = 2, value = 0.0072998753
index = 3, value = 0.0069919875
index = 4, value = 0.0067352357
index = 5, value = 0.0089222195
index = 6, value = 0.0082285808
index = 7, value = 0.0070214726
index = 8, value = 0.0054785283
index = 9, value = 0.0075605510
```

3.6 # 06. print the training (mean) accuracy over batches at last 10 epochs

```
[ ]: function_result_06()
```

```
index = 0, value = 0.9996500000
index = 1, value = 0.9996000000
```



```
index = 2, value = 0.9995500000
index = 3, value = 0.9995000000
index = 4, value = 0.9997000000
index = 5, value = 0.9990000000
index = 6, value = 0.9995000000
index = 7, value = 0.9996000000
index = 8, value = 0.9998000000
index = 9, value = 0.9993000000
```

3.7 # 07. print the testing (mean) loss over batches at last 10 epochs

```
[ ]: function_result_07()
```

```
index = 0, value = 0.0358085715
index = 1, value = 0.0335766336
index = 2, value = 0.0320852865
index = 3, value = 0.0399123657
index = 4, value = 0.0344634883
index = 5, value = 0.0377968701
index = 6, value = 0.0328307798
index = 7, value = 0.0326732362
index = 8, value = 0.0299846626
index = 9, value = 0.0307948416
```

3.8 # 08. print the testing (mean) accuracy over batches at last 10 epochs

```
[ ]: function_result_08()
```

```
index = 0, value = 0.9901250000
index = 1, value = 0.9905000000
index = 2, value = 0.9907500000
index = 3, value = 0.9890000000
index = 4, value = 0.9902500000
index = 5, value = 0.9893750000
index = 6, value = 0.9901250000
index = 7, value = 0.9907500000
index = 8, value = 0.9907500000
index = 9, value = 0.9922500000
```

3.9 # 09. print the best training (mean) accuracy within the last 10 epochs

```
[ ]: function_result_09()
```

```
best training accuracy = 0.9998000000
```

3.10 # 10. print the best testing (mean) accuracy within the last 10 epochs

```
[ ]: function_result_10()
```

```
best testing accuracy = 0.9922500000
```

```
[ ]:
```