# 计算机视觉-HW5

xTryer

2024/12/10

## 1 Task 1:Define Classfiers

### 1.1 Linear classifier

线性分类器：直接定义一个线性层，输入通道数为 in_channels, 输出通道数为 out_channels
LinearClassifier 结构如下：

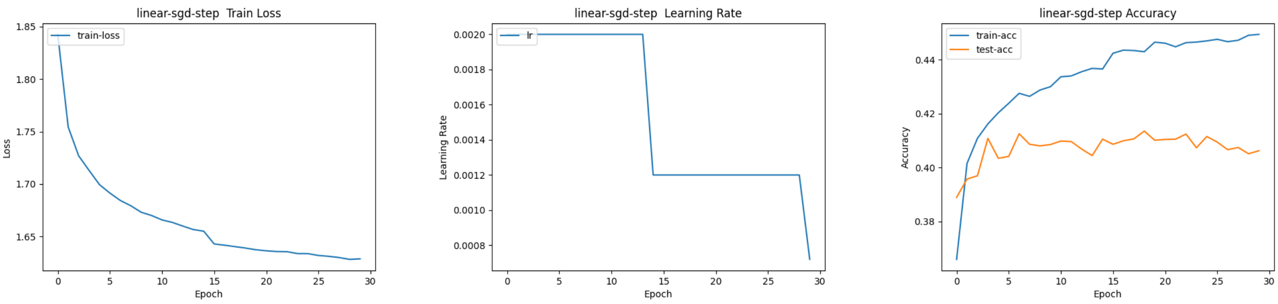
```python
class LinearClassifier(nn.Module):
  def __init__(self, in_channels: int, out_channels: int):
      super().__init__()
      self.fc1=nn.Linear(in_channels,out_channels)

  def forward(self, x: torch.Tensor):
      # flatten the input x
      y=x.view(x.size(0),-1)
      y=self.fc1(y)
      return y
```

使用 linear-SGD-StepLR 进行实验
参数设置:

- $epochnum = 30$
- $batchsize = 64$
- $SGD\_lr = 0.002$
- $SGD\_momentum = 0.8$
- $steplr\_gamma = 0.6$
- $steplr\_stepsize = 7.5$

得到结果图如下:




(a) TrainLoss　(b) LearningRate　(c) Accuracy

图 1: Task 1:Linear-SGD-StepLR

最后在 epoch 为 30 时，在训练集上的 accuracy 约为 0.41

## 1.2 Full-connected neural network classifier

全连接神经网络：考虑先经过卷积层，然后再用线性层叠加上激活函数。

FCNN 结构如下:

```
class FCNN(nn.Module):
  def __init__(self, in_channels: int, hidden_channels: int, out_channels: int):
      super().__init__()
      self.conv1=nn.Conv2d(3,30,kernel_size=5) # 30*28*28
      self.pool = nn.MaxPool2d(2,2) # 8*14*14
      self.conv2=nn.Conv2d(30,100,kernel_size=3) # 100*12*12
      # 100*6*6
      self.conv3=nn.Conv2d(100,256,kernel_size=3) # 256*4*4

      #14*5*5
      self.fc1=nn.Linear(256*4*4,80)
      self.fc2=nn.Linear(80,64)
      self.fc3=nn.Linear(64,out_channels)

      for layer in self.modules():
          if isinstance(layer,(nn.Conv2d,nn.Linear)):
              nn.init.xavier_uniform_(layer.weight)


  def forward(self, x: torch.Tensor):
      x=self.pool(F.relu(self.conv1(x)))
      x=self.pool(F.relu(self.conv2(x)))
      x=F.relu(self.conv3(x))
      x=torch.flatten(x,1)

```

```python
        x=F.relu(self.fc1(x))
        x=F.tanh(self.fc2(x))
        x=self.fc3(x)

        return x
```

# 2 Task 2:Training and Testing

dataset 与 dataload 准备:

```python
test_transform = transforms.Compose(
[transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_transform = transforms.Compose(
[transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=test_transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                          shuffle=False, num_workers=1)
```

## 2.1 Training Function

对于训练函数，我们迭代 epoch_max 次，每次迭代以 batch_size 大小遍历 trainloader, 先重置参数梯度，然后前向传递，利用交叉熵损失函数进行反向传播更新梯度，然后更新参数，每经历一个 epoch，利用 scheduler 动态调整学习率, 保存对应 epoch 的权重，并调用 test 函数在测试集上进行测试。

完整代码如下:

```python
def train(model, optimizer, scheduler, args):
criterion = nn.CrossEntropyLoss()

# for-loop
    # train
data_len = len(trainloader)
for epoch in range(epoch_max):
    running_loss=0.0
    train_acc_sum,train_num=0.0,0.0

    # get the inputs; data is a list of [inputs, labels]
    for i,data in enumerate(trainloader,0):
```

```
        inputs,labels=data
        inputs,labels=inputs.to(device),labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward
        outputs=model(inputs)

        # loss backward
        loss=criterion(outputs,labels)
        loss.backward()

        # optimize
        optimizer.step()

        running_loss+=loss.item()

        train_num +=labels.shape[0]
        train_acc_sum+=(outputs.argmax(1)==labels).sum()
    # adjust learning rate
    scheduler.step()
    # save checkpoint
    save_path = f'{local_path}/{args.model}-{args.optimizer}-{args.scheduler}/ckpt{epoch+1}.pt'
    torch.save(model,save_path)

    epoch_train_acc = train_acc_sum/train_num
    epoch_loss = running_loss/data_len
    print(f'[Epoch:{epoch+1}] loss: {epoch_loss:.3f} train-acc:{epoch_train_acc:.2f}')
    # test
    epoch_test_acc = test(model,args,epoch)
```

## 2.2 Testing Function

对于测试函数，我们直接往待评估模型中输入测试集的数据，得到其在测试集上的正确率。代码如下:

```
def test(model, args,epoch=-1):
  accurate_num,test_num=0.0,0.0
  model.eval()
  for i,data in enumerate(testloader,0):
      inputs,labels=data
```

```
        inputs,labels=inputs.to(device),labels.to(device)
        # forward
        outputs=model(inputs)

        accurate_num+=(outputs.argmax(1)==labels).sum()
        test_num+=labels.shape[0]

    test_acc = accurate_num/test_num
    if epoch==-1:
        print(f'test-acc: {test_acc:.2f}')
    else:
        print(f'[Epoch:{epoch+1}] test-acc: {test_acc:.2f}')

    return test_acc
```

# 3 Task 3:Report

除了 tensorboard 之外，我使用了 matplotlib 记录了 LearningRate,TrainLoss 和 Accuracy(train,test),

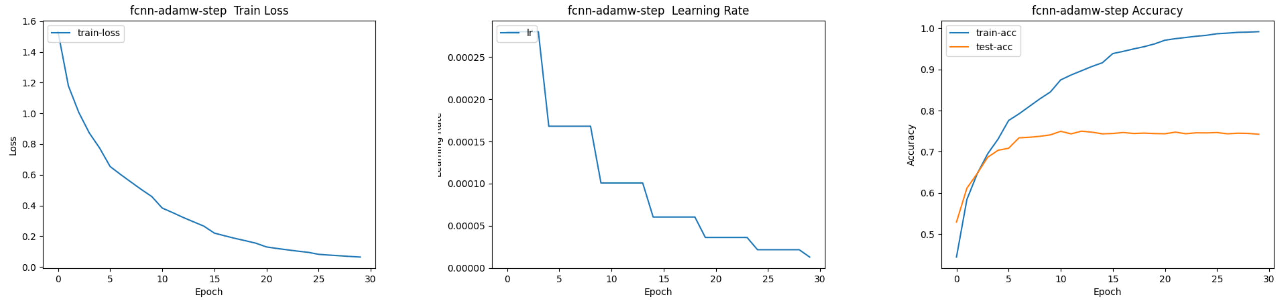
## 3.1 Compare AdamW and SGD optimizer

### 3.1.1 FCNN-AdamW-StepLR

考虑 FCNN-AdamW-StepLR,
参数设置:

- $epochnum = 30$
- $batchsize = 64$
- $AdamW\_lr = 0.00028$
- $steplr\_gamma = 0.6$
- $steplr\_stepsize = 5.0$

得到结果图如下:




(a) TrainLoss (b) LearningRate (c) Accuracy

图 2: Task 3-1:FCNN-AdamW-StepLR

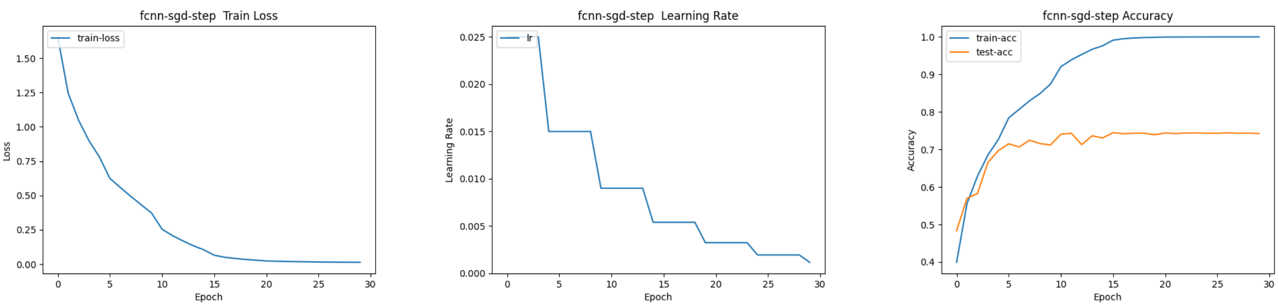
最后在 epoch 为 30 时，在训练集上的 accuracy 约为 0.74, 在 epoch 为 28 时, 在训练集上 accuracy 约为 0.75

#### 3.1.2 对于 FCNN-SGD-StepLR

考虑 FCNN-SGD-StepLR,
参数设置:

- $epochnum = 30$
- $batchsize = 64$
- $SGD\_lr = 0.025$
- $SGD\_momentum = 0.4$
- $steplr\_gamma = 0.6$
- $steplr\_stepsize = 5.0$

得到结果图如下:




(a) TrainLoss (b) LearningRate (c) Accuracy

图 3: Task 3-1:FCNN-SGD-StepLR

最后在 epoch 为 30 时，在训练集上的 accuracy 约为 0.74

### 3.1.3 Comparison

我们使用 tensorboard 可视化比较二者的 TrainLoss 曲线和 Accuracy(on testdata) 曲线

对于 TrainLoss 可视化结果如下：


图 4: Task 3-1:AdamW vs SGD-TrainLoss

对于 Accuracy(on testdata) 可视化结果如下:

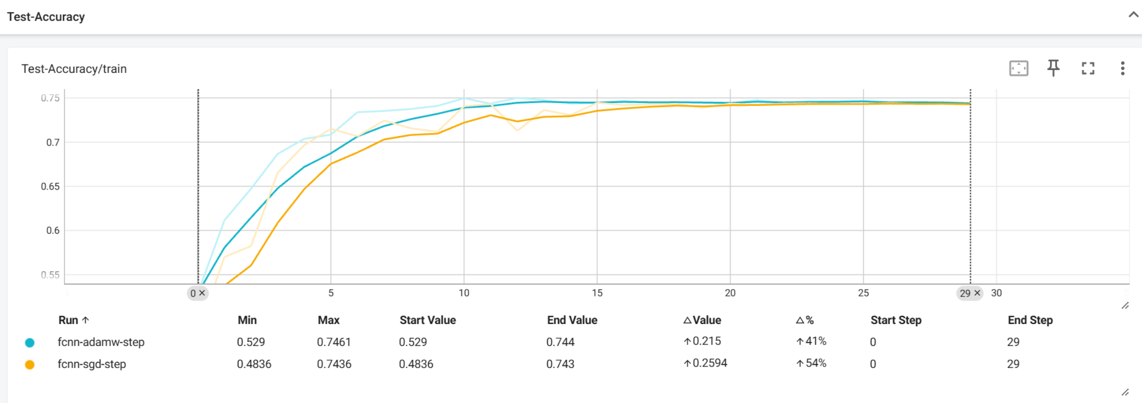



图 5: Task 3-1:AdamW vs SGD-Accuracy

根据可视化结果可以看出使用 AdamW 的模型训练过程中 trainloss 相近，在训练集上的 Accuracy 也较相近。

在这两次实验中，AdamW 的收敛速度更快, 达到最高的 test-accuracy 速度较快，训练过程中 TrainLoss 和 test-accuracy 较稳定。

而 SGD 的 Loss 变化幅度较大，test-accuracy 曲线在前期存在明显的抖动，比起 AdamW 比较不稳定。

## 3.2 Compare StepLR and CosineAnnealingLR scheduler

### 3.2.1 FCNN-AdamW-StepLR

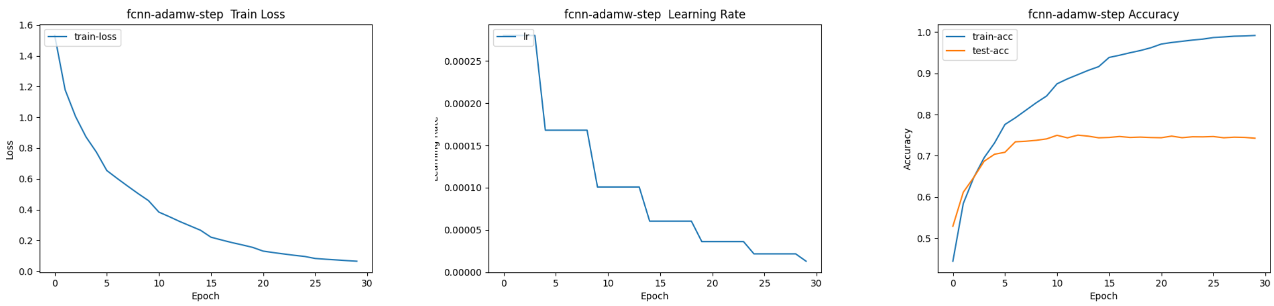
考虑 FCNN-AdamW-StepLR,

参数设置:

- $epochnum = 30$

- $batchsize = 64$
- $AdamW\_lr = 0.00028$
- $steplr\_gamma = 0.6$
- $steplr\_stepsize = 5.0$

得到结果图如下:




(a) TrainLoss (b) LearningRate (c) Accuracy

图 6: Task 3-2:FCNN-AdamW-StepLR

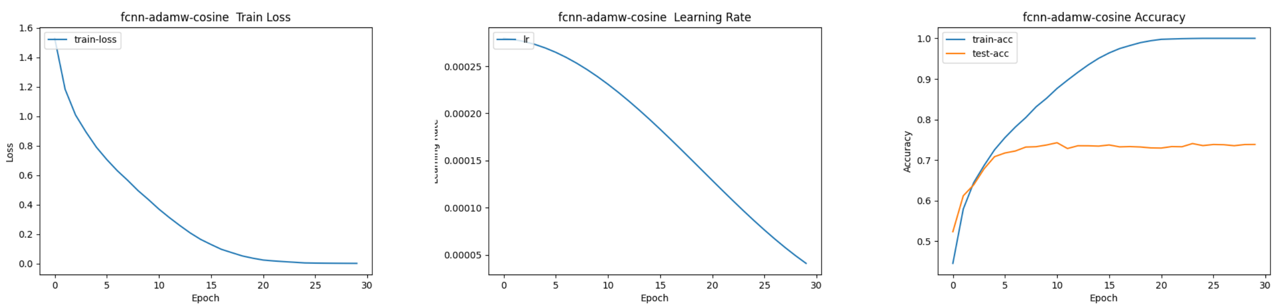
最后在 epoch 为 30 时，在训练集上的 accuracy 约为 0.74, 在 epoch 为 28 时, 在训练集上 accuracy 约为 0.75

### 3.2.2 FCNN-AdamW-CosineAnnealingLR

考虑 FCNN-AdamW-CosineAnnealingLR,
参数设置:

- $epochnum = 30$
- $batchsize = 64$
- $AdamW\_lr = 0.00028$
- $coslr\_tmaax = 40$

得到结果图如下:




(a) TrainLoss (b) LearningRate (c) Accuracy

图 7: Task 3-2:FCNN-AdamW-CosineAnnealingLR

最后在 epoch 为 30 时，在训练集上的 accuracy 约为 0.74

### 3.2.3 Comparison

我们使用 tensorboard 可视化比较二者的 TrainLoss 曲线和 Accuracy(on testdata) 曲线

对于 TrainLoss 可视化结果如下：


图 8: Task 3-2:StepLR vs CosineAnnealingLR-TrainLoss

对于 Accuracy(on testdata) 可视化结果如下:

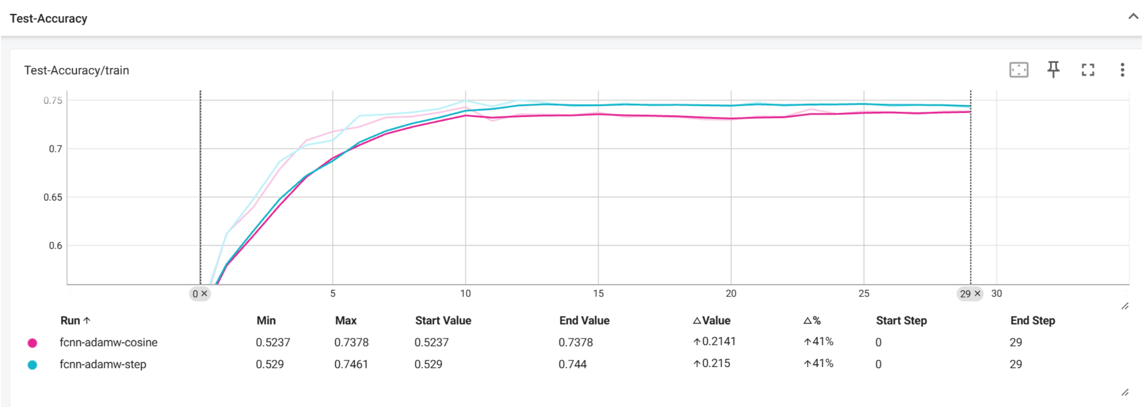


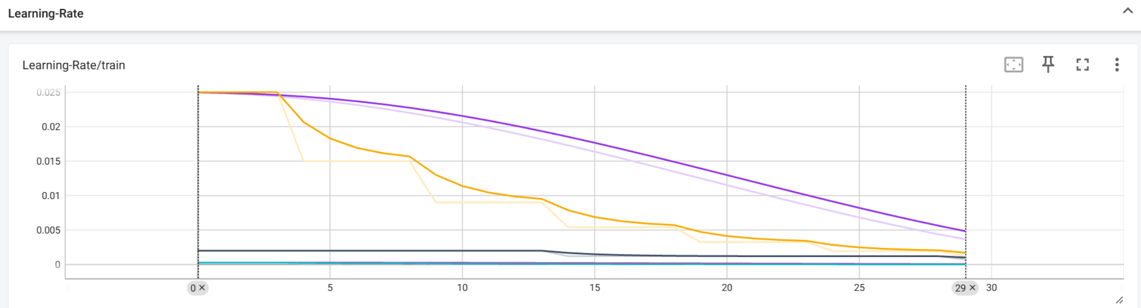

图 9: Task 3-2:StepLR vs CosineAnnealingLR-Accuracy

观察 TrainLoss 曲线，可以发现使用 CosineAnnealingLR 的模型训练 TrainLoss 更加光滑，因为其 learning_rate 是平滑调整的，而 StepLR 则是” 阶梯” 式的 (存在 lr 的突变), 二者的收敛速度差不多。

观察 Accuracy 曲线可以看出二者在训练效果上没有显著的区别, 最后达到的效果也较为相近。

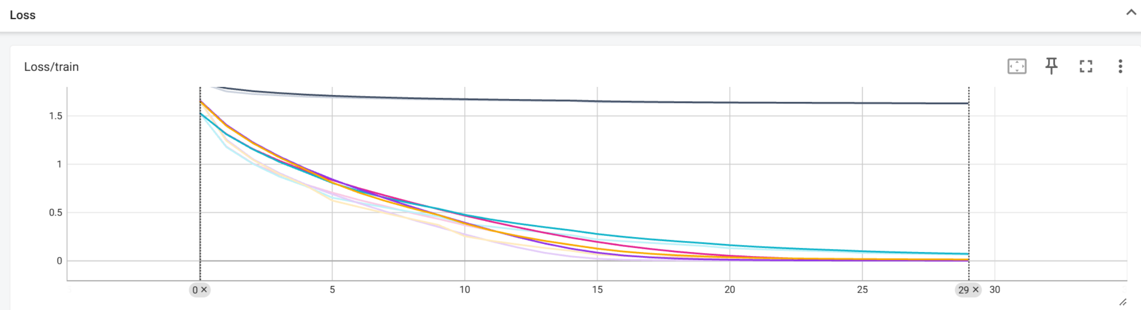
## 3.3 Visualization-total

汇总 fcnn-adamw-cosine,fcnn-adamw-step,fcnn-sgd-cosine,fcnn-sgd-step,linear-sgd-step 这五种训练结果，得到 tensorboard 结果:



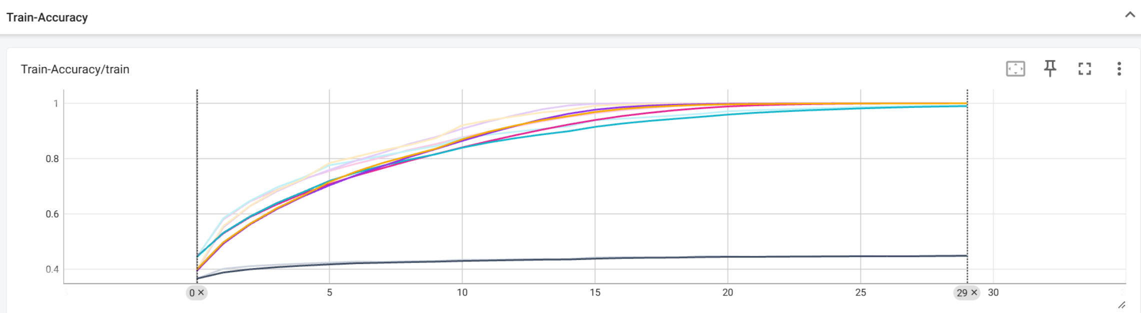

| Run ↑ | Min | Max | Start Value | End Value | ΔValue | Δ% | Start Step | End Step |
|---|---|---|---|---|---|---|---|---|
| fcnn-adamw-cosine | 0.0001 | 0.0003 | 0.0003 | 0.0001 | ↓0.0002 | ↓-81% | 0 | 29 |
| fcnn-adamw-step | 0 | 0.0003 | 0.0003 | 0 | ↓0.0003 | ↓-93% | 0 | 29 |
| fcnn-sgd-cosine | 0.0048 | 0.025 | 0.025 | 0.0048 | ↓0.0201 | ↓-81% | 0 | 29 |
| fcnn-sgd-step | 0.0017 | 0.025 | 0.025 | 0.0017 | ↓0.0233 | ↓-93% | 0 | 29 |
| linear-sgd-step | 0.001 | 0.002 | 0.002 | 0.001 | ↓0.001 | ↓-50% | 0 | 29 |

图 10: Task 3-3:Visualization-total-LearningRate



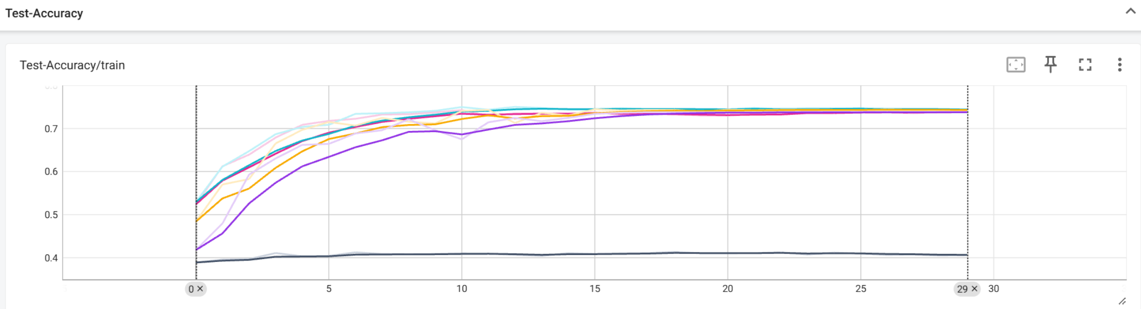

| Run ↑ | Min | Max | Start Value | End Value | ΔValue | Δ% | Start Step | End Step |
|---|---|---|---|---|---|---|---|---|
| fcnn-adamw-cosine | 0.0025 | 1.5298 | 1.5298 | 0.0025 | ↓1.5273 | ↓-100% | 0 | 29 |
| fcnn-adamw-step | 0.0729 | 1.5314 | 1.5314 | 0.0729 | ↓1.4585 | ↓-95% | 0 | 29 |
| fcnn-sgd-cosine | 0.0028 | 1.6612 | 1.6612 | 0.0028 | ↓1.6584 | ↓-100% | 0 | 29 |
| fcnn-sgd-step | 0.0146 | 1.6544 | 1.6544 | 0.0146 | ↓1.6397 | ↓-99% | 0 | 29 |
| linear-sgd-step | 1.6297 | 1.8426 | 1.8426 | 1.6297 | ↓0.2129 | ↓-12% | 0 | 29 |

图 11: Task 3-3:Visualization-total-Loss




| Run ↑ | Min | Max | Start Value | End Value | ΔValue | Δ% | Start Step | End Step |
|---|---|---|---|---|---|---|---|---|
| fcnn-adamw-cosine | 0.4456 | 0.9999 | 0.4456 | 0.9999 | ↑0.5542 | ↑124% | 0 | 29 |
| fcnn-adamw-step | 0.444 | 0.9898 | 0.444 | 0.9898 | ↑0.5458 | ↑123% | 0 | 29 |
| fcnn-sgd-cosine | 0.3939 | 1 | 0.3939 | 1 | ↑0.6061 | ↑154% | 0 | 29 |
| fcnn-sgd-step | 0.3993 | 0.9998 | 0.3993 | 0.9998 | ↑0.6005 | ↑150% | 0 | 29 |
| linear-sgd-step | 0.3659 | 0.4484 | 0.3659 | 0.4484 | ↑0.0826 | ↑23% | 0 | 29 |

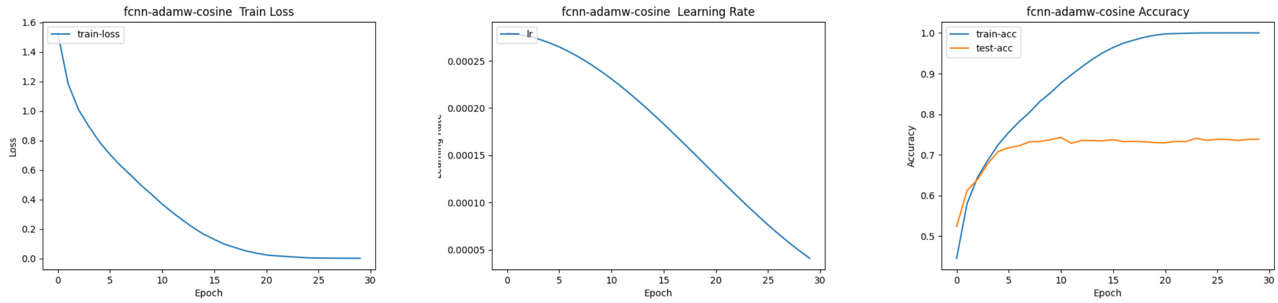
图 12: Task 3-3:Visualization-total-Accuracy on TrainData




| Run ↑ | Min | Max | Start Value | End Value | △Value | △% | Start Step | End Step |
|---|---|---|---|---|---|---|---|---|
| fcnn-adamw-cosine | 0.5237 | 0.7378 | 0.5237 | 0.7378 | ↑0.2141 | ↑41% | 0 | 29 |
| fcnn-adamw-step | 0.529 | 0.7461 | 0.529 | 0.744 | ↑0.215 | ↑41% | 0 | 29 |
| fcnn-sgd-cosine | 0.4178 | 0.7379 | 0.4178 | 0.7376 | ↑0.3198 | ↑77% | 0 | 29 |
| fcnn-sgd-step | 0.4836 | 0.7436 | 0.4836 | 0.743 | ↑0.2594 | ↑54% | 0 | 29 |
| linear-sgd-step | 0.3889 | 0.4113 | 0.3889 | 0.4066 | ↑0.0177 | ↑5% | 0 | 29 |

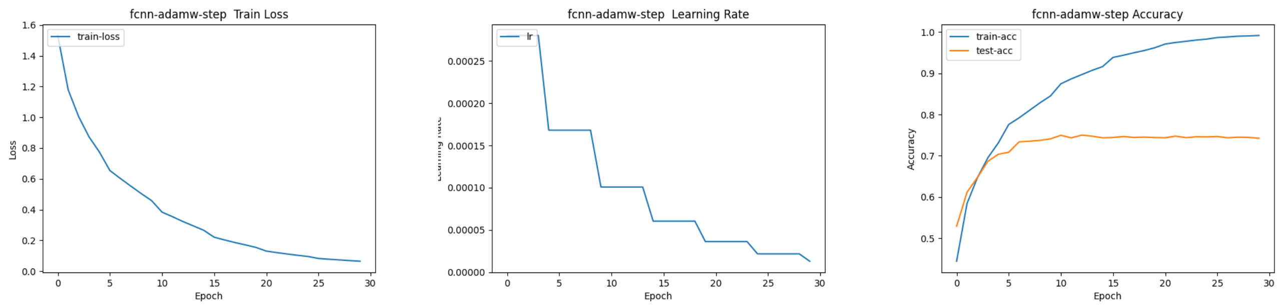
图 13: Task 3-3:Visualization-total-Accuracy on TestData

使用 Matplotlib 可视化:



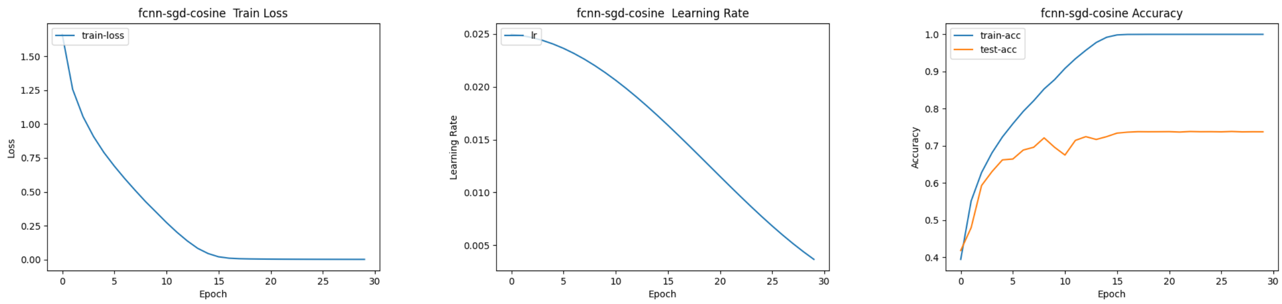

(a) TrainLoss (b) LearningRate (c) Accuracy

图 14: Task 3-3:FCNN-AdamW-CosineAnnealingLR



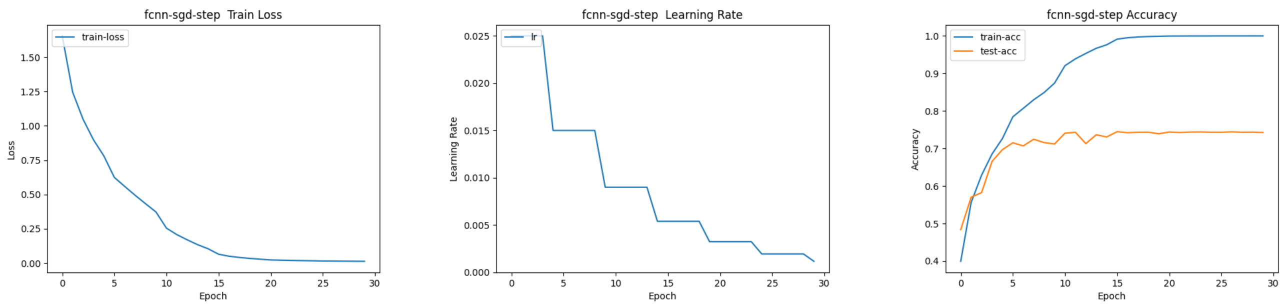

(a) TrainLoss (b) LearningRate (c) Accuracy

图 15: Task 3-3:FCNN-AdamW-StepLR



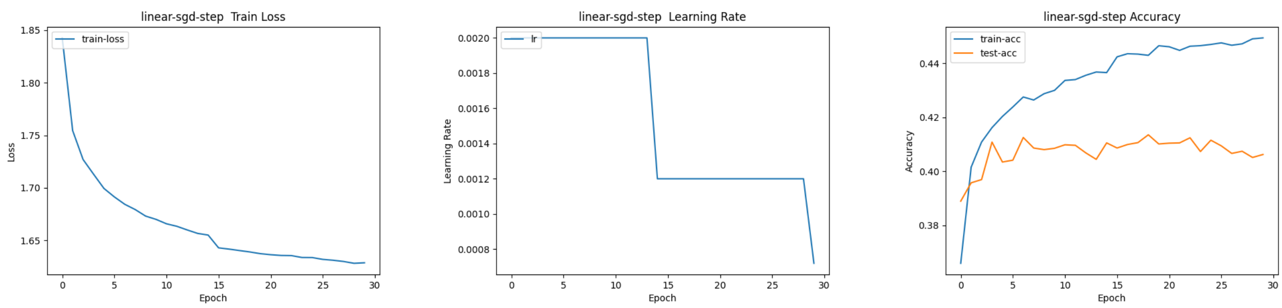

(a) TrainLoss (b) LearningRate (c) Accuracy

图 16: Task 3-3:FCNN-SGD-CosineAnnealingLR




(a) TrainLoss (b) LearningRate (c) Accuracy

图 17: Task 3-3:FCNN-SGD-StepLR




(a) TrainLoss (b) LearningRate (c) Accuracy

图 18: Task 3-3:Linear-SGD-StepLR

# 4 Task 4:A 'good' performance Model on CIFAR10

用 FCNN-AdamW-StepLR,FCNN 框架如下:

```
class FCNN(nn.Module):
def __init__(self, in_channels: int, hidden_channels: int, out_channels: int):
    super().__init__()

```

```
    self.conv1=nn.Conv2d(3,30,kernel_size=5) # 30*28*28
    self.pool = nn.MaxPool2d(2,2) # 8*14*14
    self.conv2=nn.Conv2d(30,100,kernel_size=3) # 100*12*12
    # 100*6*6
    self.conv3=nn.Conv2d(100,256,kernel_size=3) # 256*4*4

    #14*5*5
    self.fc1=nn.Linear(256*4*4,80)
    self.fc2=nn.Linear(80,64)
    self.fc3=nn.Linear(64,out_channels)

    # initialize parameters
    for layer in self.modules():
        if isinstance(layer,(nn.Conv2d,nn.Linear)):
            nn.init.xavier_uniform_(layer.weight)

def forward(self, x: torch.Tensor):
    x=self.pool(F.relu(self.conv1(x)))
    x=self.pool(F.relu(self.conv2(x)))
    x=F.relu(self.conv3(x))
    x=torch.flatten(x,1)

    x=F.relu(self.fc1(x))
    x=F.tanh(self.fc2(x))
    x=self.fc3(x)

    return x
```

参数设置:

- $epochnum = 30$
- $batchsize = 64$
- $AdamW\_lr = 0.00028$
- $steplr\_gamma = 0.6$
- $steplr\_stepsize = 5.0$

可以训练得到一个在测试集上正确率 >=0.6 的模型 (取第 28 个 epoch 得到的模型文件), 测试脚本如下:

```
import torch
import torch.nn as nn
import argparse
```

```python
import torch.nn.functional as F

import torchvision
import torchvision.transforms as transforms

import matplotlib.pyplot as plt # 可视化训练结果
import os # 文件操作

from torch.utils.tensorboard import SummaryWriter

class FCNN(nn.Module):
    # def a full-connected neural network classifier
    def __init__(self, in_channels: int, hidden_channels: int, out_channels: int):
        super().__init__()

        self.conv1=nn.Conv2d(3,30,kernel_size=5) # 30*28*28
        self.pool = nn.MaxPool2d(2,2) # 8*14*14
        self.conv2=nn.Conv2d(30,100,kernel_size=3) # 100*12*12
        # 100*6*6
        self.conv3=nn.Conv2d(100,256,kernel_size=3) # 256*4*4

        #14*5*5
        self.fc1=nn.Linear(256*4*4,80)
        self.fc2=nn.Linear(80,64)
        self.fc3=nn.Linear(64,out_channels)

        # initialize parameters
        for layer in self.modules():
            if isinstance(layer,(nn.Conv2d,nn.Linear)):
                nn.init.xavier_uniform_(layer.weight)


    def forward(self, x: torch.Tensor):
        x=self.pool(F.relu(self.conv1(x)))
        x=self.pool(F.relu(self.conv2(x)))
        x=F.relu(self.conv3(x))
        x=torch.flatten(x,1)

        x=F.relu(self.fc1(x))
        x=F.tanh(self.fc2(x))
        x=self.fc3(x)

        return x
```

```python



def test(model,epoch=-1):
    accurate_num,test_num=0.0,0.0
    model.eval()
    for i,data in enumerate(testloader,0):
        inputs,labels=data
        inputs,labels=inputs.to(device),labels.to(device)
        # forward
        outputs=model(inputs)

        accurate_num+=(outputs.argmax(1)==labels).sum()
        test_num+=labels.shape[0]

    test_acc = accurate_num/test_num
    if epoch==-1:
        print(f'test-acc: {test_acc}')
    else:
        print(f'[Epoch:{epoch+1}] test-acc: {test_acc}')

    return test_acc



if __name__ == '__main__':

    batch_size=64

    test_transform = transforms.Compose(
    [transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=test_transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              shuffle=False, num_workers=1)
    device = torch.device("cuda:0"if torch.cuda.is_available() else"cpu") # GPU

    model=torch.load("final_pt.pt")
    test(model)
```

最终输出结果:'test-acc: 0.7450999617576599', 成功!

# 5 UpLoad

- 最后的模型文件为'final_pt.pt', 可以用 test.py 测试
- runs 文件夹下包含了 tensorboard 文件
- report-data 文件夹下包含了报告中出现的所有图片以及训练日志文件