

# cv\_hw\_5

## 神经网络结构设计

### LinearClassifier

继承 `nn.Module` ,内部一个 `nn.Linear`

```
class LinearClassifier(nn.Module):
    # define a linear classifier
    def __init__(self, in_channels: int, out_channels: int):
        super().__init__()
        # in_channels: dimension of input data. For example, a RGB image [3x32x32] is converted
        # out_channels: number of categories. For CIFAR-10, it's 10
        self.fc = nn.Linear(in_channels, out_channels)

    def forward(self, x: torch.Tensor):
        x = x.view(-1, 3072)

        return self.fc(x)
```

### fcnn

内部网络用 `nn.Sequential` 整合，为使网络有更复杂的结构，这里输入`hidden_channels`为一个list，list中每个元素对应一组网络 `Linear+ReLU+Dropout` ,最后再添加一个输出`Linear`层

```

class FCNN(nn.Module):
    # def a full-connected neural network classifier
    def __init__(self, in_channels: int, hidden_channels: List[int], out_channels: int):
        super().__init__()
        # in_channels: dimension of input data. For example, a RGB image [3x32x32] is converted
        # hidden_channels
        # out_channels: number of categories. For CIFAR-10, it's 10

        # full connected layer
        # activation function
        # full connected layer
        # .....
        prev_dim = in_channels
        self.net = nn.Sequential()
        for i,hidden_dim in enumerate(hidden_channels):
            self.net.add_module('fc%d'%i, nn.Linear(prev_dim, hidden_dim))
            self.net.add_module('relu%d'%i, nn.ReLU())
            self.net.add_module('dropout%d'%i, nn.Dropout(0.1))
            prev_dim = hidden_dim

        self.net.add_module('fc%d'%(i+1), nn.Linear(prev_dim, out_channels))
    def forward(self, x: torch.Tensor):
        x = x.view(-1, 3072)

        return self.net(x)

```

## cnn

自定义了cnnclassifier继承nn.Module，内部网络分为cnnnet 和fcnet,cnnnet三

次 kernel\_size = 3,padding =1 的卷积层，中间插入两次 ReLU+ MaxPool2d(2,2) ,fcnet为一个 nn.Sequential，包含三个Linear，插入 ReLU+Dropout(0.2)

```

class CNN(nn.Module):
    # def a convolutional neural network classifier
    def __init__(self):
        super().__init__()
        self.cnnnet = nn.Sequential(
            nn.Conv2d(3,32,kernel_size = 3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(32,64,kernel_size = 3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(64,128,kernel_size = 3,padding=1),
            nn.ReLU(),
        )

        hidden_dim_1 = 128*8*8
        hidden_dim_2 = 256
        hidden_dim_3 = 96
        self.fc1 = nn.Linear(hidden_dim_1, hidden_dim_2)
        self.fc2 = nn.Linear(hidden_dim_2, hidden_dim_3)
        self.fc3 = nn.Linear(hidden_dim_3, 10)
        self.fcnet = nn.Sequential(self.fc1, nn.ReLU(), nn.Dropout(0.2), self.fc2, nn.ReLU(), nn.Linear(hidden_dim_3, 10))

    def forward(self, x: torch.Tensor):
        x = self.cnnnet(x)
        x = x.view(-1, 128*8*8)
        return self.fcnet(x)

```

# 训练与测试代码

```
def train(model, optimizer, scheduler, args):
    '''
    Model training function
    input:
        model: linear classifier or full-connected neural network classifier
        loss_function: Cross-entropy loss
        optimizer: Adamw or SGD
        scheduler: step or cosine
        args: configuration
    '''

    # create dataset
    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.train()

    # create dataloader
    trainset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
    batch_size = 128
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True, num
    #classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
    testset = datasets.CIFAR10(root='./data', train=False,
                                download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              shuffle=False, num_workers=2)

    criterion = nn.CrossEntropyLoss()
    #optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    epochs = 50

    # for-loop
    for epoch in range(epochs):
        # train
        print(f'Epoch: {epoch+1}/{epochs}')
        train_correct = 0
        train_total = 0
        l = len(trainloader)
        # get the inputs; data is a list of [inputs, labels]
```

```

for i,data in enumerate(trainloader, 0):
    inputs, labels = data
    inputs = inputs.to(device)
    labels = labels.to(device)

    # zero the parameter gradients
    optimizer.zero_grad()
    # forward
    outputs = model(inputs)
    _, predicted = torch.max(outputs.data, 1)
    train_total += labels.size(0)
    train_correct += (predicted == labels).sum().item()
    # loss backward
    loss = criterion(outputs, labels)
    loss.backward()
    # optimize
    optimizer.step()
    project.log({'loss': loss.item()},step = i+epoch*1)
print(f'Accuracy of the network on the train images:{100*train_correct / train_total}')
project.log({'train_acc': 100*train_correct / train_total},step = epoch)

# adjust learning rate
scheduler.step()

# test
# forward
# calculate accuracy
correct = 0
total = 0
with torch.no_grad():
    for i,data in enumerate(testloader,0):
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        loss = criterion(outputs, labels)
        project.log({'loss': loss.item()},step = i+epoch*1)

```

```

        print(f'Accuracy of the network on the 10000 test images:{100*correct / total} %')
    project.log({'test_acc': 100*correct / total},step = epoch)

# save checkpoint (Tutorial: https://pytorch.org/tutorials/recipes/recipes/saving\_and\_loading\_models\_for\_testing.html)
torch.save(model.state_dict(), 'checkpoint_fcnn_cos.pt')
def test(model, args):
    ...

    input:
        model: linear classifier or full-connected neural network classifier
        loss_function: Cross-entropy loss
    ...

# load checkpoint (Tutorial: https://pytorch.org/tutorials/recipes/recipes/saving\_and\_loading\_models\_for\_testing.html)
model.load_state_dict(torch.load('checkpoint_cnn.pt'))

# create testing dataset
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
testset = datasets.CIFAR10(root='./data', train=False,
                            download=True, transform=transform)

batch_size = 128
# create dataloader
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                          shuffle=False, num_workers=2)

# test
    # forward
    # calculate accuracy
correct = 0
total = 0
model.eval()
for data in testloader:
    images, labels = data

    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
print(f'Accuracy of the network on the 10000 test images:{100*correct / total} %')

```

# 可视化与对比

训练过程中的loss和accuracy变化用swanlab记录[[https://docs.swanlab.cn/guide\\_cloud/general/quick-start.html](https://docs.swanlab.cn/guide_cloud/general/quick-start.html)]

整体训练了cnn,fcnn,fcnn\_sgd,fcnn\_cos,linear,(没有后缀默认配置为adamw+step)

参数设置batch\_size = 128 ,epochs = 50

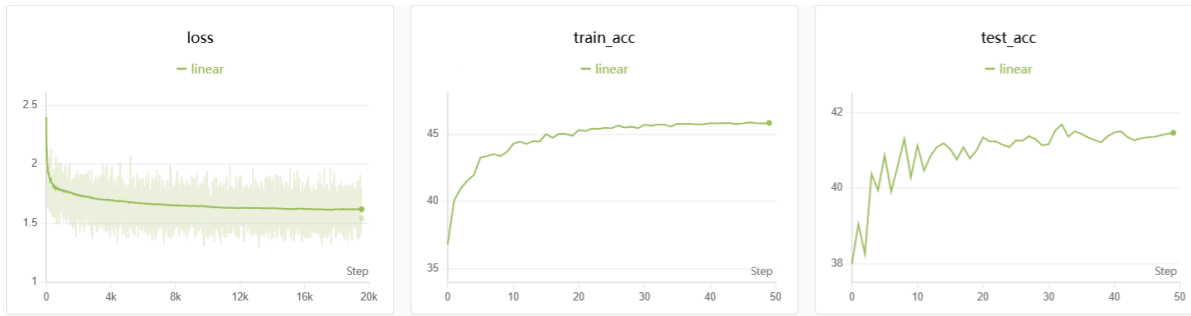
```
if args.model == 'linear':
    model = LinearClassifier(3072,10)
elif args.model == 'fcnn':
    model = FCNN(3072,[1024,512,256,128],10)
elif args.model == 'cnn':
    model = CNN()
else:
    raise AssertionError

# create optimizer
if args.optimizer == 'adamw':
    # create Adamw optimizer
    optimizer = torch.optim.AdamW(model.parameters(), lr=0.0005)
elif args.optimizer == 'sgd':
    # create SGD optimizer
    optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
else:
    raise AssertionError

# create scheduler
if args.scheduler == 'step':
    # create torch.optim.lr_scheduler.StepLR scheduler
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.6)
elif args.scheduler == 'cosine':
    # create torch.optim.lr_scheduler.CosineAnnealingLR scheduler
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)
else:
    raise AssertionError
```

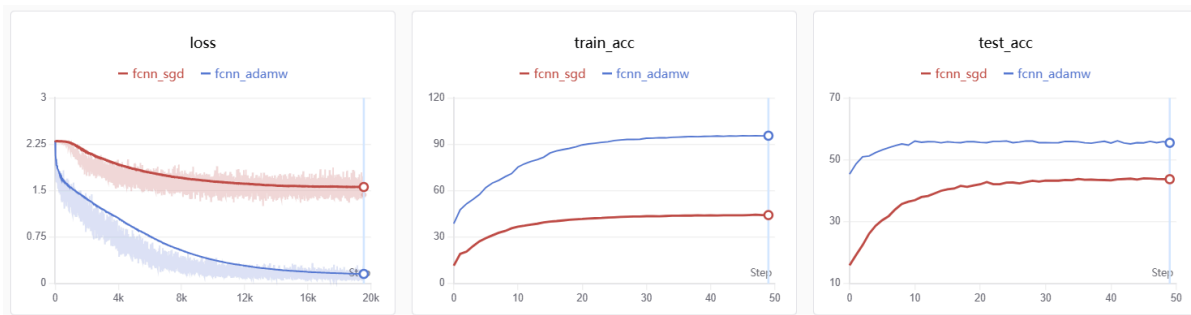
结果对比

# Linear



最终train\_acc:45.8% ， test\_acc: 41.4%

# Fcnn

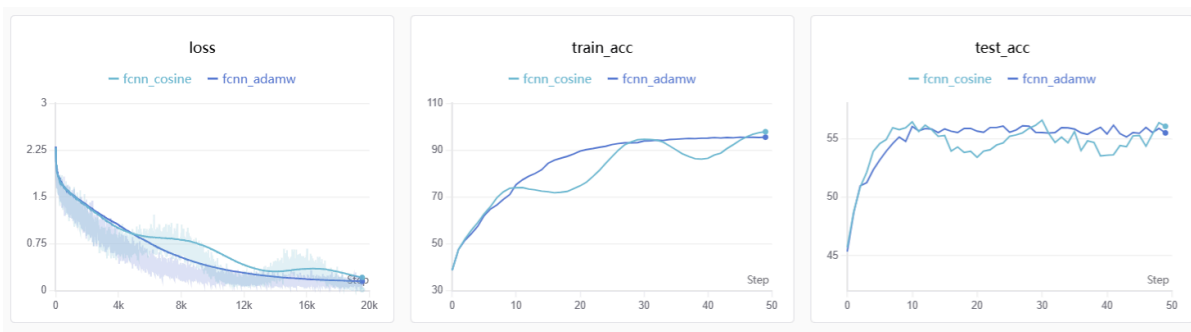


这是fcnn使用adamw和sgd优化器的对比(lr使用step)

最终50个epoch下 adamw的train\_acc: 95.5% ， test\_acc: 56.1%(epoch 41)

sgd的train\_acc: 44.2% ， test\_acc: 44%(epoch 45)

结论：adamw优化器训练的收敛速度快于sgd优化器，在给定较短epoch的训练时间下，使用adamw优化器训练可以达到更高的准确率

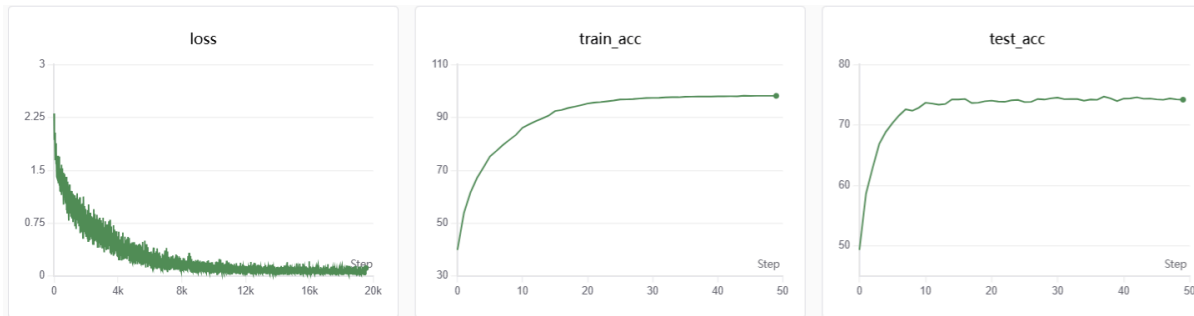


这是fcnn使用step和cosine优化器的对比(optimizer使用adamw)

最终50个epoch下，step的train\_acc: 95.5% ， test\_acc: 56.1%(epoch 41)

cosine的train\_acc: 97.8% ， test\_acc: 56.3%(epoch 48)

结论：lr scheduler使用step和cosine对比，acc上差异不大，但是使用cosine，loss的下降及acc上升明显有一个周期性效果，在每个周期末acc上升缓慢甚至小幅下降，新的周期学习率回升可能跳出局部最优



这是自定义cnn的训练结果,使用adamw和step

最终50个epoch下， train\_acc: 98.12% ， test\_acc: 76.3%

运行 `python main.py --run=test --model=cnn --optimizer=adamw --scheduler=step`

得到

**Accuracy of the network on the 10000 test images:76.3 %**

使用自定义的cnn，最终最优test——acc为 76.3%