实验一

202218018670003 马天行

00 概述

实验内容是使用手工Conv网络实现Mnist图片分类任务，文件结构等信息可以在前置文档《实验项目结构概述》中看到。

完整代码和文档见仓库：https://github.com/Noitolar/CourseDL.git

01 数据集和预处理

默认预处理为向量化与标准化，没有额外的数据增强

```
import torchvision.datasets as vdatasets
import torchvision.transforms as trans
import utils.cv as ucv


def get_dataset(config: ucv.config.ConfigObject):
    if config.dataset == "mnist":
        if config.trn_preprocess is None:
            config.trn_preprocess = trans.Compose([trans.ToTensor(), trans.Normalize(mean=[0.485], std=[0.229])])
        if config.val_preprocess is None:
            config.val_preprocess = trans.Compose([trans.ToTensor(), trans.Normalize(mean=[0.485], std=[0.229])])
        trn_set = vdatasets.MNIST(root=f"./datasets/mnist", train=True, transform=config.trn_preprocess, download=True)
        val_set = vdatasets.MNIST(root=f"./datasets/mnist", train=False, transform=config.val_preprocess, download=True)
        return trn_set, val_set
    elif ...
```

02 神经网络模型

两层网络，每一层由二维卷积+ReLU激活+BatchNorm+二维最大池化组成，卷积核大小为5*5、步长为1、填充为2、池化核大小为2*2，最终输出32个FeatureMap，每个FeatureMap大小为7*7，最终摊平经过全连接进行分类。

```
import torch
import torch.nn as nn
import torchvision.models as vmodels
```

```
class SimpleConvClassifier(nn.Module):
    def __init__(self, num_classes, num_channels):
        super().__init__()
        self.layer1 = self.build_layer(num_channels, 16)
        self.layer2 = self.build_layer(16, 32)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(32 * 7 * 7, num_classes)

    @staticmethod
    def build_layer(conv_in_channels, conv_out_channels, conv_kernel_size=5, conv_stride=1, conv_padding=2, pool_kernel_size=2):
        layer = nn.Sequential(
            nn.Conv2d(conv_in_channels, conv_out_channels, conv_kernel_size, conv_stride, conv_padding),
            nn.ReLU(), nn.BatchNorm2d(conv_out_channels), nn.MaxPool2d(pool_kernel_size))
        return layer

    def forward(self, inputs):
        outputs = self.layer1(inputs)
        outputs = self.layer2(outputs)
        outputs = self.flatten(outputs)
        outputs = self.fc(outputs)
        return outputs
```

03 模型操作

这个类用于对模型进行一些高级操作，成员包括：

- 配置项
- 记录器（用于记录训练过程中的指标以及进行日志管理）
- 设备
- 神经网络模型
- 损失函数

类方法包括：

- 日志记录：调用成员记录器的方法，将指定信息录入日志，并同步显示在终端
- 配置记录：按照固定格式将实验配置记录到日志，并同步显示在终端
- 数据转移：将内存中的张量数据转移到配置的设备的内存/显存中
- 正向调用：根据输入和标签输出预测值和损失值，如果没有输入标签则只输出预测值

在初始化成员对象之前会先设置全局随机数种子，保证实验结果可以复现。

```
import torch
import torch.nn as nn
import torchvision.transforms as trans
import time
import transformers as tfm
import utils.cv as ucv


class ModelHandlerCv(nn.Module):
    def __init__(self, config: ucv.config.ConfigObject):
        super().__init__()
        self.config = config
        tfm.set_seed(config.seed)
        self.recorder = ucv.recorder.Recorder(config.log_path)
        self.device = config.device
        self.model = config.model_class(**config.model_params).to(config.device)
        self.criterion = config.criterion_class(**config.criterion_params)

    def log(self, message):
        self.recorder.audit(message)

    def log_config(self):
        self.log(f"\n\n[+] exp starts from: {time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())}")
        for config_key, config_value in self.config.params_dict.items():
            if config_value is None:
                continue
            elif config_key.endswith("_class"):
                self.log(f"[+] {config_key.replace('_class', '')}: {config_value.__name__}")
            elif config_key.endswith("_params") and isinstance(config_value, dict):
                for param_key, param_value in config_value.items():
                    self.log(f"    [-] {config_key.replace('_params', '')}.{param_key}: {param_value}")
            elif isinstance(config_value, trans.transforms.Compose):
                self.log(f"[+] {config_key}:")
                for index, value in enumerate(str(config_value).replace(" ", "").split("\n")[1:-1]):
                    self.log(f"    [-] {index:02d}: {value}")
            else:
                self.log(f"[+] {config_key}: {config_value}")

    def device_transfer(self, data):
        if isinstance(data, torch.Tensor):
            data = data.to(self.device)
        if isinstance(data, dict):
            data = {key: value.to(self.device) for key, value in data.items()}
        return data

    def forward(self, inputs: torch.Tensor, targets=None):
        inputs = self.device_transfer(inputs)
        targets = self.device_transfer(targets)
        preds = self.model(inputs)
        loss = self.criterion(preds, targets) if targets is not None else None
        return preds, loss
```

04 记录器

这个类用于记录模型训练过程中的一些指标，以及日志管理的相关功能，成员对象包括：

- 累计准确率
- 累计损失值
- 累计样本数
- 日志记录器

成员方法包括：

- 计算并更新累计准确率、损失值、样本数
- 还原成员变量
- 返回平均准确率和损失值
- 日志录入

```
import numpy as np
import sklearn.metrics as metrics
import logging
import os


class Recorder:
    def __init__(self, logpath):
        self.accumulative_accuracy = 0.0
        self.accumulative_loss = 0.0
        self.accumulative_num_samples = 0
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.DEBUG)
        self.logger.addHandler(logging.StreamHandler(stream=None))
        if logpath is not None:
            if not os.path.exists(os.path.dirname(logpath)):
                os.makedirs(os.path.dirname(logpath))
                logfile = open(logpath, "a", encoding="utf-8")
                logfile.close()
            self.logger.addHandler(logging.FileHandler(filename=logpath, mode="a"))

    def update(self, preds, targets, loss):
        assert len(preds) == len(targets)
        num_samples = len(preds)
        preds = np.array([pred.argmax() for pred in preds.detach().cpu().numpy()])
        targets = targets.detach().cpu().numpy()
        self.accumulative_accuracy += metrics.accuracy_score(y_pred=preds, y_true=targets) * num_samples
        self.accumulative_loss += loss * num_samples
        self.accumulative_num_samples += num_samples

    def clear(self):
        self.accumulative_accuracy = 0.0
        self.accumulative_loss = 0.0
        self.accumulative_num_samples = 0

    def accuracy(self):
        accuracy = self.accumulative_accuracy / self.accumulative_num_samples
        loss = self.accumulative_loss / self.accumulative_num_samples
        return accuracy, loss

    def audit(self, msg):
        self.logger.debug(msg)
```

05 模型训练

用于训练和评估模型的类，成员对象包括：

- 模型操作器
- 优化器
- 学习率调整策略（可选）

成员方法包括：

- 训练：训练模型一轮，会调用模型操作器（ModelHandler）的记录器（Recorder）计算训练时的准确率和损失，并在本轮结束时返回训练报告并重置记录器
- 验证：验证模型，没有反向传播过程，并且不计算梯度以节省显存和算力

```
import torch
import tqdm


class Trainer:
    def __init__(self, handler):
        self.handler = handler
        self.optimizer = handler.config.optimizer_class(handler.model.parameters(),
**handler.config.optimizer_params)
```

```
        self.scheduler = handler.config.scheduler_class(self.optimizer, **handler.config.scheduler_params) if handler.config.scheduler_class is not None else None

    def train(self, loader):
        self.handler.train()
        for inputs, targets in tqdm.tqdm(loader, desc=f"    [-] training", delay=0.2, leave=False, ascii="->"):
            preds, loss = self.handler(inputs, targets)
            self.handler.recorder.update(preds, targets, loss)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
        accuracy, loss = self.handler.recorder.accuracy()
        self.handler.recorder.clear()
        if self.scheduler is not None:
            self.scheduler.step()
        report = {"loss": loss, "accuracy": accuracy}
        return report

    @torch.no_grad()
    def validate(self, loader):
        self.handler.eval()
        for inputs, targets in tqdm.tqdm(loader, desc=f"    [-] validating", delay=0.2, leave=False, ascii="->"):
            preds, loss = self.handler(inputs, targets)
            self.handler.recorder.update(preds, targets, loss)
        accuracy, loss = self.handler.recorder.accuracy()
        self.handler.recorder.clear()
        report = {"loss": loss, "accuracy": accuracy}
        return report
```

06 实验主函数

批大小为32，共训练8轮，SGD优化器初始学习率为0.001，在第4轮和第6轮下降至原来的十分之一。

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as tdata
import torchvision.transforms as trans
import utils.cv as ucv

if __name__ == "__main__":
    config = ucv.config.ConfigObject()
    config.model_class = ucv.nnmodels.SimpleConvClassifier
    config.model_params = {"num_classes": 10, "num_channels": 1}
    config.device = "cuda:0"
    config.criterion_class = nn.CrossEntropyLoss
    config.criterion_params = {}
    config.log_path = "./logs/mnist.conv.log"

    config.dataset = "mnist"
    config.seed = 0
    config.trn_preprocess = trans.Compose([trans.ToTensor(), trans.Normalize(mean=[0.485], std=[0.229])])
    config.val_preprocess = config.trn_preprocess
    config.batch_size = 32
    config.num_epochs = 8
    config.optimizer_class = optim.SGD
    config.optimizer_params = {"lr": 0.001, "momentum": 0.9, "nesterov": True}
    config.scheduler_class = optim.lr_scheduler.MultiStepLR
    config.scheduler_params = {"milestones": [4, 6], "gamma": 0.1}
    config.checkpoint_path = "./checkpoints/mnist.conv.pt"

    handler = ucv.handler.ModelHandlerCv(config)
    handler.log_config()

    trn_set, val_set = ucv.dataset.get_dataset(config)
    trn_loader = tdata.DataLoader(trn_set, batch_size=config.batch_size, shuffle=True)
    val_loader = tdata.DataLoader(val_set, batch_size=config.batch_size * 8)
    trainer = ucv.trainer.Trainer(handler)

    best_val_accuracy = 0.0
    for epoch in range(config.num_epochs):
        handler.log("    " + "=" * 40)
        trn_report = trainer.train(trn_loader)
        handler.log(f"    [{epoch + 1:03d}] trn-loss: {trn_report['loss']:.4f} --- trn-acc: {trn_report['accuracy']:.2%}")
        val_report = trainer.validate(val_loader)
        handler.log(f"    [{epoch + 1:03d}] val-loss: {val_report['loss']:.4f} --- val-acc: {val_report['accuracy']:.2%}")
        if val_report["accuracy"] > best_val_accuracy:
            best_val_accuracy = val_report["accuracy"]
            if config.checkpoint_path is not None:
                torch.save(handler.model.state_dict, config.checkpoint_path)
    handler.log(f"[=] best-val-acc: {best_val_accuracy:.2%}")
```

07 实验结果

实验日志如下，包含了本次实验的配置以及实验结果，最高验证集准确率为99.18%。

```
[+] exp starts from: 2023-04-08 23:32:41
[+] model: SimpleConvClassifier
    [-] model.num_classes: 10
    [-] model.num_channels: 1
[+] device: cuda:0
[+] criterion: CrossEntropyLoss
[+] log_path: ./logs/mnist.conv.log
[+] dataset: mnist
[+] seed: 0
[+] trn_preprocess:
    [-] 00: ToTensor()
    [-] 01: Normalize(mean=[0.485],std=[0.229])
[+] val_preprocess:
    [-] 00: ToTensor()
    [-] 01: Normalize(mean=[0.485],std=[0.229])
[+] batch_size: 32
[+] num_epochs: 8
[+] optimizer: SGD
    [-] optimizer.lr: 0.001
    [-] optimizer.momentum: 0.9
    [-] optimizer.nesterov: True
[+] scheduler: MultiStepLR
    [-] scheduler.milestones: [4, 6]
    [-] scheduler.gamma: 0.1
[+] checkpoint_path: ./checkpoints/mnist.conv.pt
    ========================================
    [001] trn-loss: 0.1085 --- trn-acc: 96.72%
    [001] val-loss: 0.0428 --- val-acc: 98.63%
    ========================================
    [002] trn-loss: 0.0394 --- trn-acc: 98.84%
    [002] val-loss: 0.0339 --- val-acc: 98.94%
    ========================================
    [003] trn-loss: 0.0271 --- trn-acc: 99.21%
    [003] val-loss: 0.0311 --- val-acc: 98.94%
    ========================================
    [004] trn-loss: 0.0196 --- trn-acc: 99.45%
    [004] val-loss: 0.0317 --- val-acc: 99.02%
    ========================================
    [005] trn-loss: 0.0118 --- trn-acc: 99.77%
    [005] val-loss: 0.0266 --- val-acc: 99.12%
    ========================================
    [006] trn-loss: 0.0103 --- trn-acc: 99.80%
    [006] val-loss: 0.0261 --- val-acc: 99.17%
    ========================================
    [007] trn-loss: 0.0095 --- trn-acc: 99.83%
    [007] val-loss: 0.0262 --- val-acc: 99.18%
    ========================================
    [008] trn-loss: 0.0096 --- trn-acc: 99.84%
    [008] val-loss: 0.0260 --- val-acc: 99.17%
[=] best-val-acc: 99.18%
```