实验四

202218018670003 马天行

00 概述

实验内容是使用手工TextCNN网络实现电影评论分类任务，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

根据数据集内容生成字典，根据课程网站上给出的中文wiki预训练word2vec模型生成可以加载到PyTorch模型中进行微调的权重矩阵。如果出现不在预训练模型中的词，就将其嵌入词向量初始化为随机分布的等长度向量。

```
import os
import tqdm
import torch
import torch.utils.data as tdata
import gensim
import numpy as np


class DatasetSentimentClassifier(tdata.Dataset):
    def __init__(self, from_file, from_vocab, sequence_length=64):
        npz_data = np.load(from_vocab, allow_pickle=True)
        self.vocab_encode = npz_data["vocab_encode"].item()
        self.sequence_length = sequence_length
        self.sentences = []
        self.targets = []
        self.load_data(from_file)

    def load_data(self, from_file):
        with open(from_file, "r", encoding="utf-8") as file:
            for line in tqdm.tqdm(file.readlines(), desc=f"[+] reading \"{from_file}\"", delay=0.2,
leave=False, ascii="->"):
                elements = line.strip().split()
                if len(elements) < 2:
                    continue
                self.targets.append(int(elements[0]))
                sentence = elements[1:]
                if len(sentence) > self.sequence_length:
                    sentence = sentence[:self.sequence_length]
```

```
                else:
                    sentence.extend(["_PAD_"] * (self.sequence_length - len(sentence)))
                self.sentences.append(sentence)

    def __getitem__(self, index):
        sentence = torch.tensor(np.array([self.vocab_encode[word] for word in self.sentences[index]]))
        target = torch.tensor(self.targets[index])
        return sentence, target

    def __len__(self):
        return len(self.sentences)

    @staticmethod
    def build_w2v(from_dir, to_file, from_pretrained_embeddings_model):
        w2v_model = gensim.models.KeyedVectors.load_word2vec_format(from_pretrained_embeddings_model,
binary=True)
        vocab_encode = {"_PAD_": 0}
        embed_size = w2v_model.vector_size
        embeddings = np.zeros(shape=(1, embed_size))
        # embeddings = np.random.uniform(-1, 1, size=(1, embed_size))
        for file_name in [name for name in os.listdir(from_dir) if name.endswith(".txt")]:
            with open(f"{from_dir}/{file_name}", "r", encoding="utf-8") as file:
                for line in tqdm.tqdm(file.readlines(), desc=f"[+] reading \"{file_name}\"", delay=0.2,
leave=False, ascii="->"):
                    for word in line.strip().split()[1:]:
                        if word not in vocab_encode.keys():
                            vocab_encode[word] = len(vocab_encode)
                            try:
                                embeddings = np.vstack([embeddings, w2v_model[word].reshape(1, embed_size)])
                            except KeyError:
                                embeddings = np.vstack([embeddings, np.random.uniform(-1, 1, size=(1,
embed_size))])
        np.savez(to_file, **{"vocab_encode": vocab_encode, "embeddings": embeddings})

    @staticmethod
    def get_embeddings_weight():
        return torch.tensor(np.load("./datasets/movie/vocab.npz", allow_pickle=True)["embeddings"],
dtype=torch.float32)
```

02 神经网络模型

模型由一至多个并行的卷积网络组成：

- 嵌入层：加载自Ginsim格式的word2vec模型，从外部配置是否冻结参数

- 卷积层：一至多个不同卷积核大小的卷积层（这里的卷积核大小仅代表第0维度，第1维度固定为嵌入层词向量长度）将输出的结果分别进行ReLU激活以及一维最大池化（池化核大小为经过卷积后的词向量长度）之后输出
- 分类层：将卷积层的一至多个输出拼接在一起后经过dropout进行全连接层做分类

```
import torch
import torch.nn as nn
import torch.nn.functional as func


class TextConvClassifier(nn.Module):
    def __init__(self, num_classes, dropout_rate, conv_out_channelses, kernel_sizes, pretrained_embeddings,
freeze_embeddings=False):
        super().__init__()
        self.embeddings = nn.Embedding.from_pretrained(pretrained_embeddings, freeze=freeze_embeddings)
        self.embed_size = int(pretrained_embeddings.shape[-1])
        self.parallel_conv_layers = nn.ModuleList([nn.Conv2d(1, conv_out_channels, (kernel_size,
self.embed_size)) for conv_out_channels, kernel_size in zip(conv_out_channelses, kernel_sizes)])
        # self.bn = nn.BatchNorm2d(conv_out_channels)
        self.dropout = nn.Dropout(dropout_rate)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(sum(conv_out_channelses), num_classes)

    def forward(self, inputs):
        outputs = self.embeddings(inputs).unsqueeze(dim=1)
        outputs = [conv_layer(outputs).squeeze(dim=3) for conv_layer in self.parallel_conv_layers]
        outputs = [func.relu(output) for output in outputs]
        outputs = [func.max_pool1d(output, output.size(dim=2)).squeeze(dim=2) for output in outputs]
        outputs = torch.cat(outputs, dim=1)
        outputs = self.dropout(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，和实验三的模型操作器继承自同一个父类，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.nlp as unlp


class ModelHandlerNLP(nn.Module):
    ...


class ModelHandlerClassifier(ModelHandlerNLP):
    def __init__(self, config: unlp.config.ConfigObject):
        super().__init__(config)

    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

第04节到第05节实验三完全一致，如果不想看可以直接跳转到第06节。

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
- 生成：仅针对NLP生成任务，根据输入的起始token以及最大输出长度输出模型生成序列，如果生成了结束符则提前终止生成

```
import torch
import tqdm
import utils.nlp as unlp


class Trainer:
    def __init__(self, handler: [unlp.handler.ModelHandlerNLP]):
        self.handler = handler
        self.config = handler.config
        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

    @torch.no_grad()
    def generate(self, input_tokens: list, output_length: int):
        self.handler.eval()
        start_token = 8291
        end_token = 8290
        if input_tokens[0] != start_token:
            input_tokens.insert(0, start_token)
        output_tokens = input_tokens
        inputs = torch.tensor(input_tokens).unsqueeze(0)
        outputs, _, hiddens = self.handler(inputs=inputs, hiddens=None)
        for _ in range(output_length - len(input_tokens)):
            preds = outputs[0][-1].argmax(axis=0)
            output_tokens.append(int(preds.item()))
            if preds.item() == end_token:
                break
            else:
                inputs = preds.reshape(1, 1)
                outputs, _, hiddens = self.handler(inputs=inputs, hiddens=hiddens)
        return output_tokens
```

06 实验主函数

配置、对象初始化、训练和评估。进行实验之前首先生成Embeddings权重矩阵保存至本地。

模型使用四种尺寸的卷积核：2、4、6、8，其对应的卷积和数量为：32、32、24、16，dropout率为0.1，允许更新来自word2vec的嵌入层模型。使用全量原始数据，最大长度为80，批大小为32，共训练4轮，AdamW优化器初始学习率为0.001，L2正则参数设为0.0001，在第二轮后学习率调整为原来的十分之一。

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

```
import torch.optim as optim
import torch.utils.data as tdata
import utils.nlp as unlp

if __name__ == "__main__":
    config = unlp.config.ConfigObject()

    config.model_class = unlp.nnmodels.TextConvClassifier
    config.model_params = {"num_classes": 2, "dropout_rate": 0.1, "kernel_sizes": [2, 4, 6, 8],
"conv_out_channelses": [32, 32, 24, 16],
                           "freeze_embeddings": False, "pretrained_embeddings":
unlp.dataset.DatasetSentimentClassifier.get_embeddings_weight()}
    config.device = "cuda:0"
    config.criterion_class = nn.CrossEntropyLoss
    config.criterion_params = {}
    config.log_path = "./logs/movie.conv.log"

    config.dataset = "movie"
    config.seed = 0
    config.sequence_length = 80
    config.batch_size = 32
    config.num_epochs = 4
    config.optimizer_class = optim.AdamW
    config.optimizer_params = {"lr": 0.001, "weight_decay": 1e-4}
    config.scheduler_class = optim.lr_scheduler.StepLR
    config.scheduler_params = {"step_size": 2, "gamma": 0.1}
    config.checkpoint_path = "./checkpoints/movie.conv.pt"

    handler = unlp.handler.ModelHandlerClassifier(config)
    handler.log_config()

    # unlp.dataset.DatasetSentimentClassifier.build_w2v(from_dir="./datasets/movie",
to_file="./datasets/movie/vocab.npz",
from_pretrained_embeddings_model="./datasets/movie/wiki_word2vec_50.bin")
    trn_set = unlp.dataset.DatasetSentimentClassifier(from_file="./datasets/movie/train.txt",
from_vocab="./datasets/movie/vocab.npz", sequence_length=config.sequence_length)
    val_set = unlp.dataset.DatasetSentimentClassifier(from_file="./datasets/movie/validation.txt",
from_vocab="./datasets/movie/vocab.npz", sequence_length=config.sequence_length)
    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 = unlp.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85.10%。

```
[+] exp starts from: 2023-04-16 20:22:06
[+] model: TextConvClassifier
    [-] model.num_classes: 2
    [-] model.dropout_rate: 0.1
    [-] model.kernel_sizes: [2, 4, 6, 8]
    [-] model.conv_out_channelses: [32, 32, 24, 16]
    [-] model.freeze_embeddings: False
    [-] model.pretrained_embeddings: tensor([
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.1642, -0.1085,  0.1950,  ..., -0.4986,  0.2594, -0.2626],
        [ 0.3688, -0.1484,  0.0068,  ..., -0.5649,  0.3491, -0.5529],
        ...,
        [-0.3737, -0.5502, -0.2598,  ...,  0.4632,  0.4255,  0.0466],
        [ 0.2875,  0.2226,  0.6066,  ..., -0.1017,  0.1671,  0.1913],
        [ 0.5021, -0.4564,  0.2155,  ..., -0.3437,  0.1961, -0.0354]])
[+] device: cuda:0
[+] criterion: CrossEntropyLoss
[+] log_path: ./logs/movie.conv.log
[+] dataset: movie
[+] seed: 0
[+] sequence_length: 80
[+] batch_size: 32
[+] num_epochs: 4
[+] optimizer: AdamW
    [-] optimizer.lr: 0.001
    [-] optimizer.weight_decay: 0.0001
[+] scheduler: StepLR
    [-] scheduler.step_size: 2
    [-] scheduler.gamma: 0.1
[+] checkpoint_path: ./checkpoints/movie.conv.pt
```

```
    ======================================
    [001] trn-loss: 0.4782 --- trn-acc: 76.64%
    [001] val-loss: 0.4220 --- val-acc: 80.78%
    ======================================
    [002] trn-loss: 0.2956 --- trn-acc: 87.64%
    [002] val-loss: 0.3614 --- val-acc: 84.79%
    ======================================
    [003] trn-loss: 0.1672 --- trn-acc: 94.45%
    [003] val-loss: 0.3645 --- val-acc: 85.10%
    ======================================
    [004] trn-loss: 0.1507 --- trn-acc: 94.93%
    [004] val-loss: 0.3716 --- val-acc: 85.01%
[=] best-val-acc: 85.10%
```