实验三

202218018670003 马天行

00 概述

实验内容是使用手工LSTM网络实现唐诗生成任务，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

课程网站上给出的原始数据是空格填充在前，实际诗句在后，总长度125。个人认为这样的数据不利于训练循环神经网络，所及将空格放在后面，并且设置了最大序列长度（默认为50）保证不会出现文字和空格填充之间没有结束符的情况。同时由于提供的原属数据太大，这里也设置了数据使用量（默认全量使用）来加速训练过程。同时数据集对象还有token的编解码功能。

数据集每次取数据（假设序列长度为N）的前N-1个token作为模型输入，后N-1个token作为模型标签。

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


class DatasetPoemGenerator(tdata.Dataset):
    def __init__(self, sequence_length=50, use_samples=-1):
        npz_data = np.load(f"./datasets/poem/tang.npz", allow_pickle=True)
        self.vocab = {"encode": npz_data["word2ix"].item(), "decode": npz_data["ix2word"].item()}
        if use_samples == -1:
            self.sentences = npz_data["data"]
        else:
            self.sentences = npz_data["data"][:use_samples]
        self.sequence_length = sequence_length
        self.preprocess()

    def preprocess(self):
        new_sentences = []
        for sentence in self.sentences:
            new_sentence = [token for token in sentence if token != 8292]
            if len(new_sentence) < self.sequence_length:
                new_sentence.extend([8292] * (self.sequence_length - len(new_sentence)))
```

```
            else:
                new_sentence = new_sentence[:self.sequence_length]
            new_sentences.append(new_sentence)
        self.sentences = np.array(new_sentences)
        self.sentences = torch.tensor(self.sentences, dtype=torch.long)

    def encode(self, character: str):
        return self.vocab["encode"][character]

    def decode(self, token: int):
        return self.vocab["decode"][token]

    def __getitem__(self, index):
        sentence = self.sentences[index, :-1]
        target = self.sentences[index, 1:]
        return sentence, target

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

02 神经网络模型

模型为一个多层LSTM结构，具体结构为;

- 嵌入层：输入维度就是词典长度，输出维度由外部配置
- LSTM层：由外部配置决定层数以及层与层之间的dropout率（在Windows下使用带有dropout的多层LSTM会产生CUDA告警，在StackOverflow上发现这个可能是Torch当前版本存在的问题）
- 分类层：由两层全连接组成，中间加了一个Tahn激活

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


class LstmGnerator(nn.Module):
    def __init__(self, vocab_size, embed_size, lstm_output_size, num_lstm_layers=1, lstm_dropout=0.0):
        super().__init__()
        self.lstm_output_size = lstm_output_size
        self.num_lstm_layers = num_lstm_layers
        # 在windows上多层lstm使用dropout或导致driver shutdown告警，应该是torch的问题
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, lstm_output_size, num_layers=num_lstm_layers, batch_first=True,
dropout=lstm_dropout)
        self.fc = nn.Sequential(
            nn.Linear(lstm_output_size, 2048),
```

```
            nn.Tanh(),
            nn.Linear(2048, vocab_size))

    def forward(self, inputs, hiddens):
        outputs = self.embedding(inputs)
        # lstm_outputs.shape: (batch_size, sequence_length, vocab_size)
        # lstm_hiddens: (lstm_h0, lstm_c0)
        outputs, lstm_hiddens = self.lstm(outputs, hiddens)
        outputs = self.fc(outputs)
        return outputs, lstm_hiddens
```

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、模型隐藏值（可选）和标签（可选）输出预测值、模型隐藏值和损失值，如果没有输入模型隐藏值则将其初始化为等大小同设备的全零张量，如果没有输入标签则只输出预测值和模型隐藏值

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):
    def __init__(self, config: unlp.config.ConfigObject):
        super().__init__()
        self.config = config
        tfm.set_seed(config.seed)
```

```
        self.recorder = unlp.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_key.endswith("_class"):
                self.log(f"[+] {config_key.replace('_class', '')}: {config_value.__name__ if config_value is
not None else None}")
            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()}
        if isinstance(data, tuple):
            data = tuple([child.to(self.device) for child in data])
        if isinstance(data, list):
            data = [child.to(self.device) for child in data]
        return data


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

    def forward(self, inputs: torch.Tensor, hiddens: tuple = None, targets: torch.Tensor = None):
        if hiddens is None:
            batch_size = inputs.shape[0]
            lstm_h0 = torch.zeros(size=(self.model.num_lstm_layers, batch_size, self.model.lstm_output_size),
dtype=torch.float, requires_grad=False)
```

```
            lstm_c0 = torch.zeros(size=(self.model.num_lstm_layers, batch_size, self.model.lstm_output_size),
dtype=torch.float, requires_grad=False)
            hiddens = (lstm_h0, lstm_c0)
        inputs = self.device_transfer(inputs)
        targets = self.device_transfer(targets)
        hiddens = self.device_transfer(hiddens)
        preds, hiddens = self.model(inputs, hiddens)

        # 相当于把batch内的多个样本拼接起来算损失函数
        if targets is not None:
            batch_size, sequence_length, vocab_size = preds.shape
            preds = preds.reshape(batch_size * sequence_length, vocab_size)
            targets = targets.reshape(batch_size * sequence_length)
            loss = self.criterion(preds, targets)
            self.recorder.update(preds, targets, loss)
        else:
            loss = None
        return preds, loss, hiddens
```

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
- 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。

模型的嵌入层输出特征为512维，隐藏值为1024维，使用3层LSTM，层与层之间的dropout率为50%。使用原始数据中的前640条，最大长度为50，批大小为32，共训练20轮，AdamW优化器初始学习率为0.002，L2正则参数设为0.0001，不额外调整学习率。每次通过让模型生成以“风”开头的序列来评估模型质量。

```
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.LstmGnerator
    config.model_params = {"vocab_size": 8293, "lstm_input_size": 512, "lstm_output_size": 1024,
"num_lstm_layers": 3, "lstm_dropout": 0.5}
    config.device = "cuda:0"
    config.criterion_class = nn.CrossEntropyLoss
    config.criterion_params = {}
    config.dataset_class = unlp.dataset.DatasetPoemGenerator
    config.dataset_params = {"sequence_length": 50, "use_samples": 640}
```

```
    config.log_path = f"./logs/poem({config.dataset_params['use_samples']}).lstm.log"
    config.seed = 0
    config.batch_size = 32
    config.num_epochs = 20
    config.optimizer_class = optim.AdamW
    config.optimizer_params = {"lr": 0.002, "weight_decay": 1e-4}
    config.scheduler_class = None
    config.checkpoint_path = f"./checkpoints/poem({config.dataset_params['use_samples']}).lstm.pt"

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

    dataset = config.dataset_class(**config.dataset_params)
    trn_loader = tdata.DataLoader(dataset, batch_size=config.batch_size, shuffle=True)
    trainer = unlp.trainer.Trainer(handler)

    best_accuracy = 0.0
    best_generation = ""
    for index in range(config.num_epochs):
        handler.log("    " + "=" * 40)
        report = trainer.train(trn_loader, index)
        tokens = trainer.generate(input_tokens=[dataset.encode(x) for x in "风"], output_length=50)
        generation_sample = "".join(dataset.decode(x) for x in tokens)
        handler.log(f"    [{index + 1:03d}] {generation_sample}")
        if report["accuracy"] > best_accuracy:
            best_accuracy = report["accuracy"]
            best_generation = generation_sample
            if config.checkpoint_path is not None:
                torch.save(handler.model.state_dict, config.checkpoint_path)
    handler.log(f"[=] best-acc: {best_accuracy:.2%}")
    handler.log(f"[=] best-generation: {best_generation}")
```

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，最高训练准确率时生成内容为：“风阁何馆霜，散宫散幽林。高阁霭新节，高光洒华襟。徒髯趋芳席，终与閒人魂。”

```
[+] exp starts from: 2023-04-08 23:56:52
[+] model: LstmGnerator
    [-] model.vocab_size: 8293
    [-] model.lstm_input_size: 512
    [-] model.lstm_output_size: 1024
    [-] model.num_lstm_layers: 3
    [-] model.lstm_dropout: 0.5
[+] device: cuda:0
```

```
[+] criterion: CrossEntropyLoss
[+] dataset: DatasetPoemGenerator
    [-] dataset.sequence_length: 50
    [-] dataset.use_samples: 640
[+] log_path: ./logs/poem(640).lstm.log
[+] seed: 0
[+] batch_size: 32
[+] num_epochs: 20
[+] optimizer: AdamW
    [-] optimizer.lr: 0.002
    [-] optimizer.weight_decay: 0.0001
[+] scheduler: None
[+] checkpoint_path: ./checkpoints/poem(640).lstm.pt
    ========================================
    [001] trn-loss: 7.3294 --- trn-acc: 11.80%
    [001] <START>
风, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,
    ========================================
    [002] trn-loss: 5.6176 --- trn-acc: 22.90%
    [002] <START>风山<EOP>
    ========================================
    [003] trn-loss: 5.3750 --- trn-acc: 24.41%
    [003] <START>风山不不，山山山山。<EOP>
    ========================================
    [004] trn-loss: 5.2521 --- trn-acc: 24.57%
    [004] <START>风山不山，山人不中。<EOP>
    ========================================
    [005] trn-loss: 5.1551 --- trn-acc: 24.60%
    [005] <START>风人一山，山人一山。<EOP>
    ========================================
    [006] trn-loss: 5.0807 --- trn-acc: 24.71%
    [006] <START>风年不不远，不山一山时。<EOP>
    ========================================
    [007] trn-loss: 5.0085 --- trn-acc: 24.67%
    [007] <START>风年不已郡，晨山复自秋。<EOP>
    ========================================
    [008] trn-loss: 4.9615 --- trn-acc: 24.79%
    [008] <START>风子一已子，春山不不秋。<EOP>
    ========================================
    [009] trn-loss: 4.8971 --- trn-acc: 24.87%
    [009] <START>风山一山人，山山一山门。<EOP>
    ========================================
    [010] trn-loss: 4.8190 --- trn-acc: 24.94%
    [010] <START>风月生云凉，晨是独中行。<EOP>
    ========================================
    [011] trn-loss: 4.7500 --- trn-acc: 25.11%
```

[011] <START>风年不已郡，西人独幽时。<EOP>

==========

[012] trn-loss: 4.6528 --- trn-acc: 25.32%

[012] <START>风藩不已久，幽林亦未欣。还来无已远，高此独未施。<EOP>

==========

[013] trn-loss: 4.5453 --- trn-acc: 25.54%

[013] <START>风年欲山去，山山出幽门。还从方所攀，高来一所疎。<EOP>

==========

[014] trn-loss: 4.4194 --- trn-acc: 25.86%

[014] <START>风子滴云兮岐边之，一年青人一炜然。立人不可脱琼去，欲有先人满人薛。<EOP>

==========

[015] trn-loss: 4.3017 --- trn-acc: 26.41%

[015] <START>风子寒云度，西山在幽里。始见无芳里，高月亦已同。始见心已永，高是已幽情。<EOP>

==========

[016] trn-loss: 4.1723 --- trn-acc: 26.85%

[016] <START>风树春城暮，晨木亦相同。还来何人动，高树夜南川。端居无相见，高人何南眠。<EOP>

==========

[017] trn-loss: 4.0198 --- trn-acc: 27.71%

[017] <START>风年郡郡郡，青月已清襟。还见南海散，还是清城曙。还见南海散，还见清城曲。<EOP>

==========

[018] trn-loss: 3.8670 --- trn-acc: 28.71%

[018] <START>风阁非京构，晨咏一雾襟。还君飘所职，欲复独纷持。还然须海巅，高书复归前。还怀飘园气，高书复归归。<EOP>

==========

[019] trn-loss: 3.6855 --- trn-acc: 30.17%

[019] <START>风阁澄芳燕，西为已伊门。诸门已已永，高里已华曙。还怀故园郡，独见此田时。<EOP>

==========

[020] trn-loss: 3.5179 --- trn-acc: 31.45%

[020] <START>风阁何馆霜，散宫散幽林。高阁霭新节，高光洒华襟。徒髯趁芳席，终与閒人魂。<EOP>

[=] best-acc: 31.45%

[=] best-generation: <START>风阁何馆霜，散宫散幽林。高阁霭新节，高光洒华襟。徒髯趁芳席，终与閒人魂。<EOP>