LSTM

lesi chen

基于char-RNN的唐诗生成，数据集来自网上，下载链接附在代码中。

首先在data.py中预处理jason格式的唐诗，并且按照七言和五言分成不同的数据类别储存。在每句诗的前后添加'<START' 以及 '<END' 表示诗句的开始和结束。

```
#保存唐诗
data7 = []
data5 = []
for i in tqdm.tqdm(range(20)):
    path = "./json/poet.tang." + str(i*1000) + ".json"
    with open("./json/poet.song.0.json","rb") as f:
        jsondata = json.loads(f.read())
        for poetry in jsondata:
            p = poetry.get("paragraphs")
            for s in p:
                if s == "":
                    continue
                ss = re.split('。',s)
                for w in ss:
                    if len(w) == 15 and w[7] == ', ':
                        data7.append(list(w))
                    elif len(w) == 11 and w[5] == ', ':
                        data5.append(list(w))

def vec2poem(s,ix2word):
    ns = ""
    for x in s:
        if x != "</s>" and x != "<END>" and x != "<STRAT>":
            ns = ns + ix2word[x]
    return ns

def processData(data,path):
    words = {_word for _sentence in data for _word in _sentence}
    word2ix = {_word: _ix for _ix, _word in enumerate(words)}
    word2ix['<END>'] = len(word2ix)
    word2ix['<START>'] = len(word2ix)
    word2ix['</s>'] = len(word2ix)
    ix2word = {_ix: _word for _word, _ix in list(word2ix.items())}

    for i in range(len(data)):
```

```
        data[i] = ["<START>"] + data[i] + ["<END>"]

    data = [[word2ix[_word] for _word in _sentence] for _sentence in data]

    print(len(data))
    print(vec2poem(data[2],ix2word))
    print(vec2poem(data[3],ix2word))

    np.save(path + "data.npy", data)
    np.save(path + "ix2word.npy", ix2word)
    np.save(path + "word2ix.npy", word2ix)

processData(data7, "./datasets/tang7/")
processData(data5, "./datasets/tang5/")
```

利用Embedding和LSTM定义诗歌生成模型并训练。

```
# REF: https://github.com/chenyuntc/pytorch-book/blob/master/chapter09-neural_poet_RNN/model.py

device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")

class PoetryModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=2)
        self.linear = nn.Linear(hidden_dim, vocab_size)

    def forward(self, inputs, hidden=None):
        seq_len, batch_size = inputs.size()
        if hidden is None:
            h0 = inputs.data.new(2, batch_size, self.hidden_dim).fill_(0).float()
            c0 = inputs.data.new(2, batch_size, self.hidden_dim).fill_(0).float()
        else:
            h0, c0 = hidden

        embeds = self.embeddings(inputs)
        output, hidden = self.lstm(embeds, (h0, c0))
        output = self.linear(output.view(seq_len * batch_size, -1))

        return output, hidden

class PoetryGererater():
    def __init__(self):
        self.word2ix = np.load("./datasets/tang5/word2ix.npy", allow_pickle=True).item()
        self.ix2word = np.load("./datasets/tang5/ix2word.npy", allow_pickle=True).item()

        #print(self.word2ix['平'])
        #print(self.word2ix['安'])
        #print(self.word2ix['復'])
        #print(self.word2ix['旦'])

        self.model = PoetryModel(len(self.word2ix), 128, 256)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-3)
```

```
        self.criterion = nn.CrossEntropyLoss()
        self.epoch = 20

    def train(self):
        data = np.load("./datasets/tang5/data.npy", allow_pickle=True)
        data = torch.from_numpy(data)
        data = data.to(device)
        dataloader = torch.utils.data.DataLoader(data,batch_size=128,shuffle=True)

        self.model.to(device)

        LossList = []
        for epoch in tqdm.tqdm(range(self.epoch)):
            L = 0
            for i, data in enumerate(dataloader):
                data = data.long().transpose(1, 0).contiguous()
                self.optimizer.zero_grad()
                inputs, target = data[:-1, :], data[1:, :]
                output, _ = self.model(inputs)
                loss = self.criterion(output, target.view(-1))
                L += loss.item()
                loss.backward()
                self.optimizer.step()
            LossList.append(L)
            if epoch % 1 == 0:
                print(self.generate())

        plt.plot(LossList)
        plt.savefig("./dump/loss.png")
        #plt.show()

    def generate(self,start_words="平安復旦"):

        results = ""


        for sw in start_words:
            hidden = None
            inputs = torch.LongTensor([self.word2ix['<START>']]).view(1, 1).to(device)
            output, hidden = self.model(inputs, hidden)

            sentence = ""
            w = str(sw)
            for i in range(12):
                sentence += w
                inputs = torch.LongTensor([self.word2ix[w]]).view(1, 1).to(device)
                output, hidden = self.model(inputs, hidden)
                top_index = output.data[0].topk(1)[1][0].item()
                w = self.ix2word[top_index]
            results += sentence
        return results

def main(training):
    PG = PoetryGererater()
    if training is True:
        PG.train()
        torch.save(PG.model.state_dict(), "./checkpoints/model5.pth")
    else:
        PG.model.load_state_dict(torch.load("./checkpoints/model5.pth"))
        print(PG.generate())
```

```
if __name__ == "__main__":
    main(training=True)
```

生成藏头诗的结果如下所示，

平生榮事爲先'生，一曲金魚是我非

安得故人長葉影，石雲深處水清東

復山高水鎖烟微，風雨前僧到中行

旦來高齋出遊才，不知才看道有家

平生初望最知在，一念閒名信不尊

安開別墅偏天子，還見萬途偏自分

復山花雨暮烟霞，一徑春高別有聲

旦來南國歸難字，白髮新客在人間

平生榮遇更誰如，竊位妨賢四紀餘

安得故人頻會面，一罇相對共醺酣

復嶺春多多病詔，聯鑣萬名最如依

旦山烟月酒臨河，萬種新霞照紫老

平生心氣在，終在靜邊塵

安民即是道，投足皆爲家

復川尋北知，風日照孤綸

旦英共攀折，芳歲幾推移