**实验三 自动写诗**

1. 实验目的
2. 理解和掌握循环神经网络概念及在深度学习框架中的实现。
3. 掌握使用深度学习框架进行文本生成任务的基本流程：如数据读取、构造网络、训练和预测等。
4. 实验要求
5. 基于Python语言和任意一种深度学习框架（实验指导书中使用Pytorch框架进行介绍），完成数据读取、网络设计、网络构建、模型训练和模型测试等过程，最终实现一个可以自动写诗的程序。网络结构设计要有自己的方案，不能与实验指导书完全相同。
6. 随意给出首句，如给定“湖光秋月两相和”，输出模型续写的诗句。也可以根据自己的兴趣，进一步实现写藏头诗（不做要求）。要求输出的诗句尽可能地满足汉语语法和表达习惯。实验提供预处理后的唐诗数据集，包含57580首唐诗（在课程网站下载），也可以使用其他唐诗数据集。
7. 按规定时间在课程网站提交实验报告、代码以及PPT。
8. 实验原理

实验使用到的网络结构主要有Embedding、LSTM以及全连接层。在pytorch中分别定义如下：

1. Class torch.nn.Embedding(num\_embeddings, embedding\_dim, padding\_idx=None, max\_norm=None, norm\_type=2.0, scale\_grad\_by\_freq=False, sparse=False, \_weight=None)

参数介绍：

num\_embeddings: 词汇表的大小。

embedding\_dim: 每个嵌入向量的维度。

padding\_idx: 如果提供的话，输出遇到此下标时用零填充。

max\_norm: 如果提供的话，会对词嵌入进行归一化，使它们的范数小于提供的值。

norm\_type: 对于max\_norm选项计算p范数时的p。

scale\_grad\_by\_freq: 如果提供的话，会根据mini-batch中单词频率缩放梯度。

1. Class torch.nn.LSTM(input\_size, hidden\_size, num\_layers=1, bias=True, batch\_first=False, dropout=0, bidirectional=False)

参数介绍：

input\_size: 输入的特征数目。

hidden\_size: 隐状态的特征数目。

num\_layers: LSTM的层数。

bias:是否使用偏置。

batch\_first: 是否将batch放在第一维。如果为True，那么输入输出的Tensor形式都是(batch, seq\_len, num\_features)。

dropout: 如果非零，则除了最后一层，每层的输出都加个Dropout层。

bidirectional: 是否使用双向LSTM。

输入：(注: 以下均考虑batch\_first为False)

intput, 形式为(seq\_len, batch, input\_size)。

h\_0, 形式为(num\_layers \* num\_directions, batch, hidden\_size)。

c\_0, 形式为(num\_layers \* num\_directions, batch, hidden\_size)。

输出：

output，形式为(seq\_len, batch, num\_directions \* hidden\_size)。

h\_n, 形式为(num\_layers \* num\_directions, batch, hidden\_size)。

c\_n, 形式为(num\_layers \* num\_directions, batch, hidden\_size)。

1. Class torch.nn.Linear(in\_features, out\_features, bias=True)。

参数：

in\_features: 每个输入样本的大小。

out\_features: 每个输出样本的大小。

bias: 默认值True，若设置为False，这层不会学习偏置。

四、 实验所用数据集及工具

实验提供预处理过的数据集，含有57580首唐诗，每首诗限定在125词，不足125词的以</s>填充。数据集以npz文件形式保存，包含三个部分：

1. data: 诗词数据，将诗词中的字转化为其在字典中的序号表示。
2. ix2word: 序号到字的映射
3. word2ix: 字到序号的映射

除了此数据集，也可自由选择其他唐诗数据集进行训练，根据需要自行预处理，如：<https://github.com/chinese-poetry/chinese-poetry>。

五、实验步骤与方法

实验可简单地划分为数据准备、模型构建、模型训练和模型预测四个步骤。

1. 数据准备

def prepareData():

datas = np.load("tang.npz")

data = datas['data']

ix2word = datas['ix2word'].item()

word2ix = datas['word2ix'].item()

data = torch.from\_numpy(data)

dataloader = DataLoader(data,

batch\_size=16,

shuffle=True,

num\_workers=2)

return dataloader, ix2word, word2ix

1. 模型构建

模型构建时，需要创建一个继承自nn.Module的类，在这个类的\_init\_( )中定义网络结构，在forward中定义前向传播过程。

示例中的模型相当简单，只有Embedding、单层LSTM、全连接层三个网络层，同学们可以自行进行拓展，以获得更好的效果。

示例：

class PoetryModel(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_dim):

super(PoetryModel, self).\_\_init\_\_()

self.hidden\_dim = hidden\_dim

self.embeddings = nn.Embedding(vocab\_size, embedding\_dim)

self.lstm = nn.LSTM(embedding\_dim, self.hidden\_dim, num\_layers=1, batch\_first=True)

self.linear = nn.Linear(self.hidden\_dim, vocab\_size)

def forward(self, input, hidden=None):

embeds = self.embeddings(input) # [batch, seq\_len] => [batch, seq\_len, embed\_dim]

batch\_size, seq\_len = input.size()

if hidden is None:

h\_0 = input.data.new(1, batch\_size, self.hidden\_dim).fill\_(0).float()

c\_0 = input.data.new(1, batch\_size, self.hidden\_dim).fill\_(0).float()

else:

h\_0, c\_0 = hidden

output, hidden = self.lstm(embeds, (h\_0, c\_0))

output = self.linear(output)

output = output.reshape(batch\_size \* seq\_len, -1)

return output, hidden

1. 模型训练

模型训练包括定义模型、设置优化器和损失函数、获取模型输出、计算误差、误差反向传播等步骤。

def train(dataloader, word2ix):

# 定义模型

model = PoetryModel(len(word2ix),

embedding\_dim=Config.embedding\_dim,

hidden\_dim=Config.hidden\_dim)

optimizer = optim.Adam(model.parameters(), lr=Config.lr)

criterion = nn.CrossEntropyLoss()

loss\_meter = meter.AverageValueMeter()

1. 模型预测

def generate(model, start\_words, ix2word, word2ix):

results = list(start\_words)

start\_words\_len = len(start\_words)

# 第一个词语是<START>

input = t.Tensor([word2ix['<START>']]).view(1, 1).long()

hidden = None

model.eval()

with torch.no\_grad():

for i in range(Config.max\_gen\_len):

output, hidden = model(input, hidden)

# 如果在给定的句首中，input为句首中的下一个字

if i < start\_words\_len:

w = results[i]

input = input.data.new([word2ix[w]]).view(1, 1)

# 否则将output作为下一个input进行

else:

top\_index = output.data[0].topk(1)[1][0].item()

w = ix2word[top\_index]

results.append(w)

input = input.data.new([top\_index]).view(1, 1)

if w == '<EOP>':

del results[-1]

break

return results