5、训练脚本的搭建

import random

import time

import torch

import torch.nn as nn

from torch import optim

from datasets import readLangs, SOS\_token, EOS\_token, MAX\_LENGTH

from models import EncoderRNN, AttenDecoderRNN

from utils import timeSince

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

MAX\_LENGTH += 1 # 添加了终止符,比dataset中的的最大长度多1，因为要加入终止符

# 本任务完成英文到中文的翻译。若要倒过来，则要修改lang1和lang2的位置，还有pairs中的中英文词样本对的位置

lang1 = "en"

lang2 = "cn"

path = "../data/cmn.txt"

input\_lang, output\_lang, pairs = readLangs(lang1, lang2, path)

# print(len(pairs))

# print(input\_lang.n\_words)

# print(input\_lang.index2word)

# print(output\_lang.n\_words)

# print(output\_lang.index2word)

def listTotensor(input\_lang, data):

indexes\_in = [input\_lang.word2index[word] for word in data.split(" ")] #得到句子所对应的索引列表[3,6,3,...]，经过embedding层，变为二维向量

indexes\_in.append(EOS\_token) # 在最后加入终止符,所以要比dataset中得MAX\_LENGTH大1

input\_tensor = torch.tensor(indexes\_in,

dtype=torch.long,

device=device).view(-1, 1)

return input\_tensor # 转换为张量并输出

#把pairs下的序列转换为输入tensor，并在tensor中插入一个终止符

# 将一个样本对转化为tensor

def tensorsFromPair(pair):

input\_tensor = listTotensor(input\_lang, pair[0]) # 将样本对前半部分英文转化为索引列表

output\_tensor = listTotensor(output\_lang, pair[1]) # 将样本对后半部分中文转化为索引列表

return (input\_tensor, output\_tensor)

# 计算loss

def loss\_func(input\_tensor, output\_tensor, encoder, decoder, encoder\_optimizer, decoder\_optimizer,criterion):

encoder\_hidden = encoder.initHidden() #初始化隐藏层

encoder\_optimizer.zero\_grad() #优化器梯度置零

decoder\_optimizer.zero\_grad()

input\_len = input\_tensor.size(0) # 输入输出长度，input\_tensor,output\_tensor均为二维张量。# 一句话的长度，

output\_len = output\_tensor.size(0) # input\_tensor.size(1):为一个词的表示维度(embedding层的输出大小)

encoder\_outputs = torch.zeros(MAX\_LENGTH, encoder.hidden\_size, device=device) # encoder的输出

#每次从input\_tensor中取一个出来利用隐藏层信息进行encoder

for ei in range(input\_len): # 将一个一句话的每个词依次编码

encoder\_output, encoder\_hidden = encoder(input\_tensor[ei], encoder\_hidden)

encoder\_outputs[ei] = encoder\_output[0, 0] #编码结果， # encoder\_output为3维的向量

# encoder\_outputs为一个句子的编码结果，为二维张量[[],[]...]

# 定义解码器

decoder\_hidden = encoder\_hidden

decoder\_input = torch.tensor([[SOS\_token]], device=device) #第一个解码输入定义为起始符SOS\_token

# 加入随机因子，随机修改当前隐藏层的输入为真实的label，让模型收敛更快

use\_teacher\_forcing = True if random.random() < 0.5 else False

loss = 0 #loss初始化为0

if use\_teacher\_forcing: # 满足条件，使用

for di in range(output\_len):

decoder\_output, decoder\_hidden, decoder\_attention = decoder(

decoder\_input, decoder\_hidden, encoder\_outputs

) # encoder\_outputs:要解码的内容

loss += criterion(decoder\_output, output\_tensor[di]) # 计算loss, output\_tensor:期待的输出(也就是label)

decoder\_input = output\_tensor[di] #下一次循环的输入直接定义为真实的label

else:

for di in range(output\_len): # 不满足条件

decoder\_output, decoder\_hidden, decoder\_attention = decoder(

decoder\_input, decoder\_hidden, encoder\_outputs

)

loss += criterion(decoder\_output, output\_tensor[di])

# 定义下一次的输入为当前的预测结果

topV, topi = decoder\_output.topk(1)

decoder\_input = topi.squeeze().detach()

# 判断解码是否结束

if decoder\_input.item() == EOS\_token: # 等于终止符，解码结束

break

loss.backward() #梯度传播

encoder\_optimizer.step()

decoder\_optimizer.step()

return loss.item() / output\_len

######

# 定义网络

hidden\_size = 256

encoder = EncoderRNN(input\_lang.n\_words, hidden\_size).to(device)

decoder = AttenDecoderRNN(hidden\_size, output\_lang.n\_words,

max\_len = MAX\_LENGTH,

dropout\_p=0.1).to(device)

lr = 0.01

encoder\_optimizer = optim.SGD(encoder.parameters(), lr=lr) # 编码器优化器

decoder\_optimizer = optim.SGD(decoder.parameters(), lr=lr) # 解码器优化器

#设置学习率调整 # 学习率的调整策略

scheduler\_encoder = torch.optim.lr\_scheduler.StepLR(encoder\_optimizer,

step\_size=1,

gamma=0.95)

scheduler\_decoder = torch.optim.lr\_scheduler.StepLR(decoder\_optimizer,

step\_size=1,

gamma=0.95)

# 定义损失函数

criterion = nn.NLLLoss()

# 不使用dataset,dataloader

# 直接生成样本对训练

n\_iters = 10000 # 最大迭代次数

training\_pairs = [

tensorsFromPair(random.choice(pairs)) for i in range(n\_iters) # 挑选1000000个样本对

]

print\_every = 1000 # 每迭代1000词打印一次信息

save\_every = 10000

print\_loss\_total = 0

start = time.time()

for iter in range(1, n\_iters+1):

training\_pair = training\_pairs[iter - 1]

input\_tensor = training\_pair[0]

output\_tensor = training\_pair[1]

loss = loss\_func(input\_tensor,

output\_tensor,

encoder,

decoder,

encoder\_optimizer,

decoder\_optimizer,

criterion)

print\_loss\_total += loss

if iter % print\_every == 0:

print\_loss\_avg = print\_loss\_total / print\_every

print\_loss\_total = 0

print("{},{},{},{}".format(timeSince(start, iter/n\_iters),

iter, iter / n\_iters \* 100,

print\_loss\_avg))

#保存模型

if iter % save\_every == 0:

torch.save(encoder.state\_dict(),

"../models/encoder\_{}.pth".format(iter))

torch.save(decoder.state\_dict(),

"../models/decoder\_{}.pth".format(iter))

#更新学习率

if iter % 1000:

scheduler\_encoder.step()

scheduler\_decoder.step()

6、测试脚本的搭建

"""

利用训练好的模型进行推理计算

复用train.py的代码

去掉loss\_func、学习率和优化器等部分代码

加载已经训练好的参数

"""

import random

import torch

import torch.nn as nn

from torch import optim

from datasets import readLangs, SOS\_token, EOS\_token, MAX\_LENGTH

from models import EncoderRNN, AttenDecoderRNN

from utils import timeSince

import time

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

MAX\_LENGTH = MAX\_LENGTH + 1

lang1 = "en"

lang2 = "cn"

path = "../data/cmn.txt"

input\_lang, output\_lang, pairs = readLangs(lang1, lang2, path)

print(len(pairs))

print(input\_lang.n\_words)

print(input\_lang.index2word)

print(output\_lang.n\_words)

print(output\_lang.index2word)

def listTotensor(input\_lang, data):

indexes\_in = [input\_lang.word2index[word] for word in data.split(" ")]

indexes\_in.append(EOS\_token)

input\_tensor = torch.tensor(indexes\_in,

dtype=torch.long,

device=device).view(-1, 1)

return input\_tensor

def tensorsFromPair(pair):

input\_tensor = listTotensor(input\_lang, pair[0])

output\_tensor = listTotensor(output\_lang, pair[1])

return (input\_tensor, output\_tensor)

hidden\_size = 256

encoder = EncoderRNN(input\_lang.n\_words, hidden\_size).to(device)

decoder = AttenDecoderRNN(hidden\_size,

output\_lang.n\_words,

max\_len=MAX\_LENGTH,

dropout\_p=0.1).to(device)

# 加载已经训练好的参数

encoder.load\_state\_dict(torch.load("../models/encoder\_10000.pth"))

decoder.load\_state\_dict(torch.load("../models/decoder\_10000.pth"))

n\_iters = 10

train\_sen\_pairs = [

random.choice(pairs) for i in range(n\_iters)

]

training\_pairs = [

tensorsFromPair(train\_sen\_pairs[i]) for i in range(n\_iters)

]

for i in range(n\_iters):

input\_tensor, output\_tensor = training\_pairs[i]

encoder\_hidden = encoder.initHidden()

input\_len = input\_tensor.size(0)

encoder\_outputs = torch.zeros(MAX\_LENGTH, encoder.hidden\_size, device=device)

for ei in range(input\_len):

encoder\_output, encoder\_hidden = encoder(input\_tensor[ei], encoder\_hidden)

encoder\_outputs[ei] = encoder\_output[0, 0]

decoder\_hidden = encoder\_hidden

decoder\_input = torch.tensor([[SOS\_token]], device=device)

use\_teacher\_forcing = True if random.random() < 0.5 else False

decoder\_words = []

for di in range(MAX\_LENGTH):

decoder\_output, decoder\_hidden, decoder\_attention = decoder(

decoder\_input, decoder\_hidden, encoder\_outputs

)

topV, topi = decoder\_output.topk(1)

decoder\_input = topi.squeeze().detach()

# 如果预测结果==终止符

if topi.item() == EOS\_token: # 加入终止符

decoder\_words.append("<EOS>")

break

else: # 加入预测结果

decoder\_words.append(output\_lang.index2word[topi.item()])

print(train\_sen\_pairs[i][0]) # input

print(train\_sen\_pairs[i][1]) # output

print(decoder\_words)

AI写代码

python

运行