分数：	

深度学习

2020 - 2021 学年度第 二 学期

实践报告

任课教师：孔雨秋

题目：循环神经网络实现歌词生成（RNN GRU LSTM）

院(系)：人工智能学院

班级：电智 1902

学号：201981498

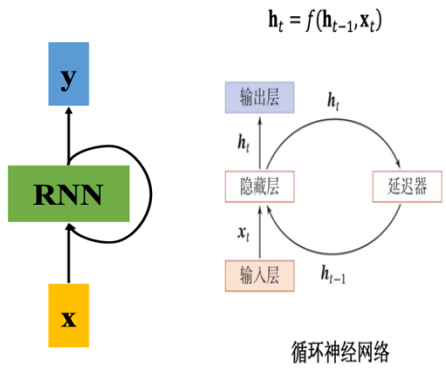
姓名：李铭潍

任务：RNN 实现歌词生成

1. 理论基础

(1). 循环神经网络 RNN

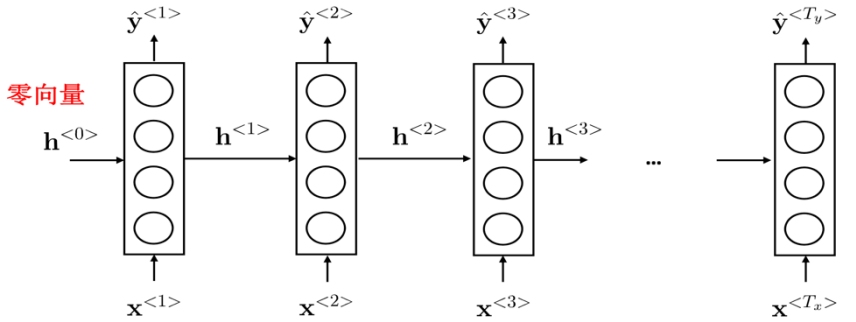
循环神经网络(RNN)是基于记忆的网络模型，期望网络能够记住前面出现的特征，并依据特征推断出后面的结果，整体网络结构不断循环，因此称为循环神经网络。



循环神经网络

RNN 能够将网络的输出保存在一个记忆单元格中，
这个记忆单元与下一次的输入一起进入神经网络中。

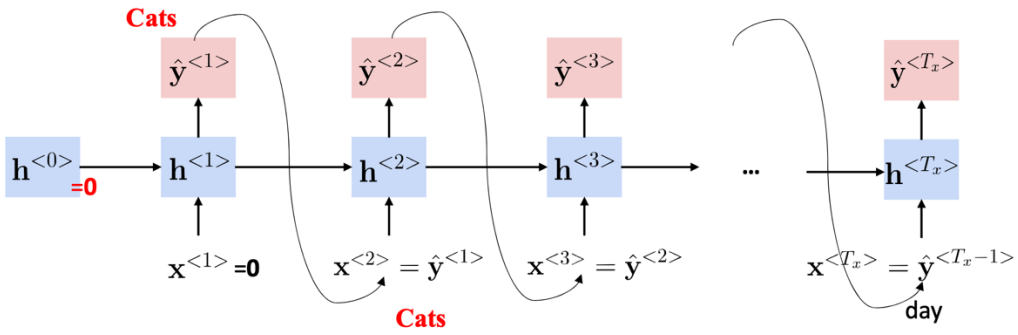
可以看到，时刻 i 输入为 X，输出为 Y，
时刻［0,i-1］区间的状态信息被反馈至网络中。
将序列中的每个数据点依次输入，
Xt-1 进入隐藏层，输出的结果 ht-1 保存中单元格中，
作为记忆单元与 Xt 一起进入神经网络。



RNN 是单方向的

序列不断输入神经网络
从左到右扫描序列数据
每个时间共享参数

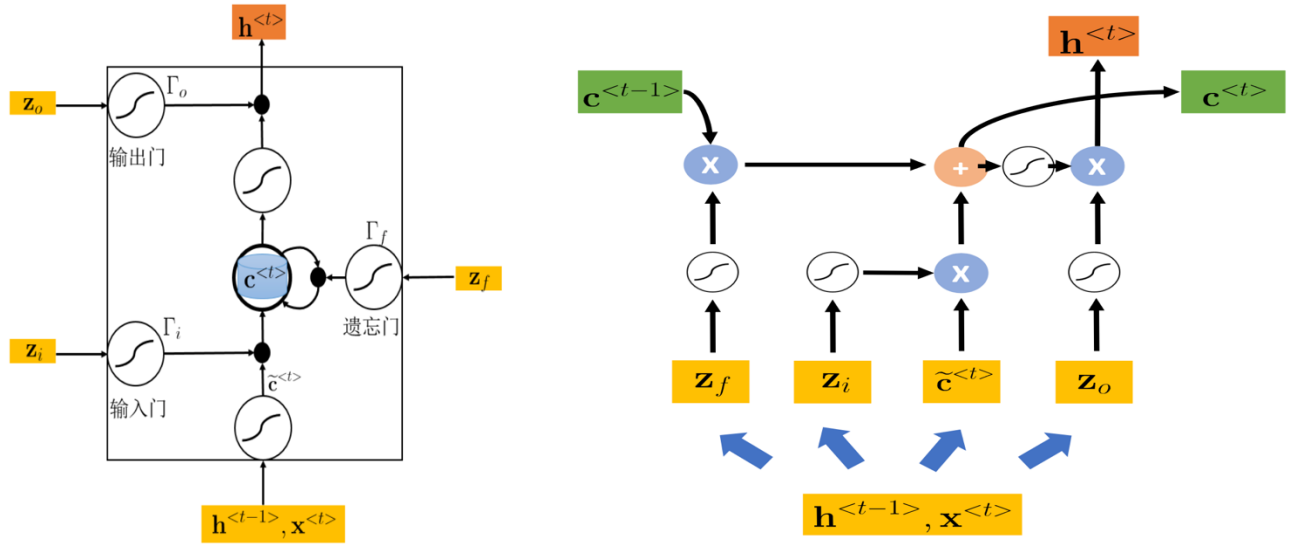
利用 RNN 训练语言模型



1. 在每个时间步下，根据输出的概率分布进行采样
2. 将采样得到的单词作为下一个时间步的输入，前个输入的字符作为下个输出字符预测依据

(2). LSTM (Long Short Term Memory Networks)

长短时记忆网络可以控制何时让输入进入神经元，何时记住之前时序中学到的东西，以及何时让输出传递到下一个时间戳。主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。



LSTM 参数比标准 RNN 多，维度是标准 RNN 维度 4 倍，因为 LSTM 中间比标准 RNN 多了三个线性变换，多的线性变换权重拼接在一起。LSTM 里做了 4 个类似标准 RNN 运算，所有参数个数也是标准 RNN 的 4 倍。

LSTM 的输入隐藏状态除了 ht 还有 Ct。他们合在一起成为网络的隐藏状态，而且他们大小完全一样，将数据输入 LSTM 后：

首先决定从单元格中舍弃哪些信息，由遗忘门的 Sigmoid 层实现，查看 ht-1 和 xt，并未单元格状态 Ct-1 中的每一个数字输出 0 和 1 之间的数字。1 表会完全保留，0 表示彻底删除。

下一步是确定要添加到单元状态的信息，有两个部分：

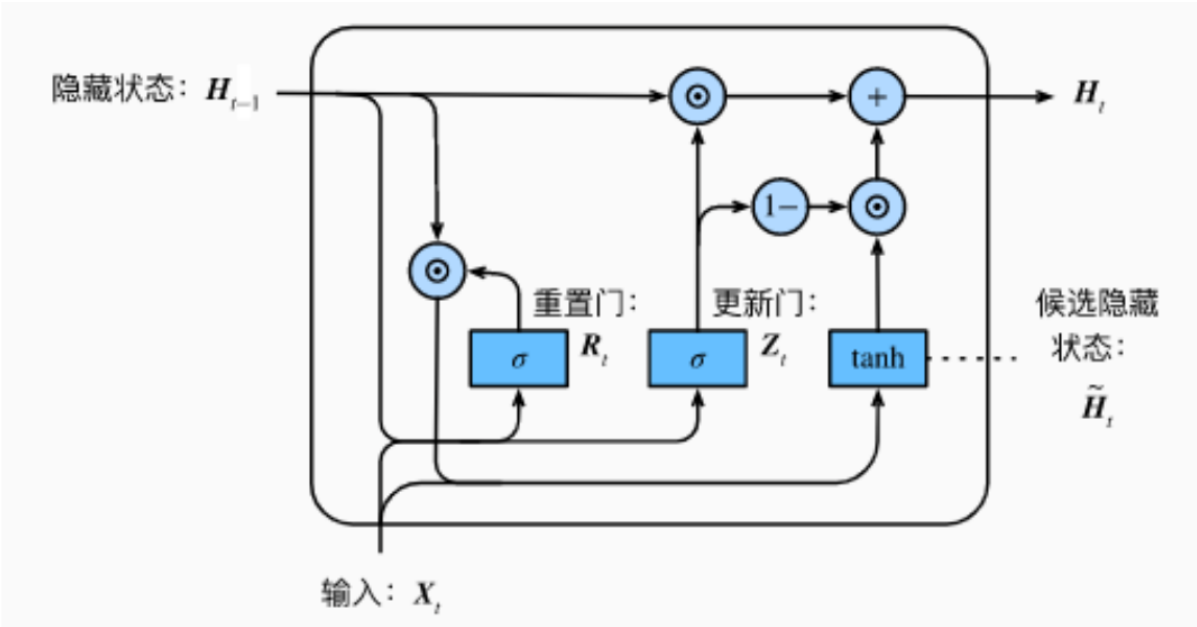
一个是 Sigmoid 层，称为输入层，决定要更新的值；

另一个是 tanh 层，创建要添加到单元格状态的新值。

下一步，将输入门的 Sigmoid 层与 tanh 函数生成的两个值组合在一起，通过在遗忘门和 it 以及 Ct 的乘积之和之间进行逐元素乘法更新单元状态。

最后，决定输出。

(3). GRU



GRU 对 LSTM 做了两个改动：

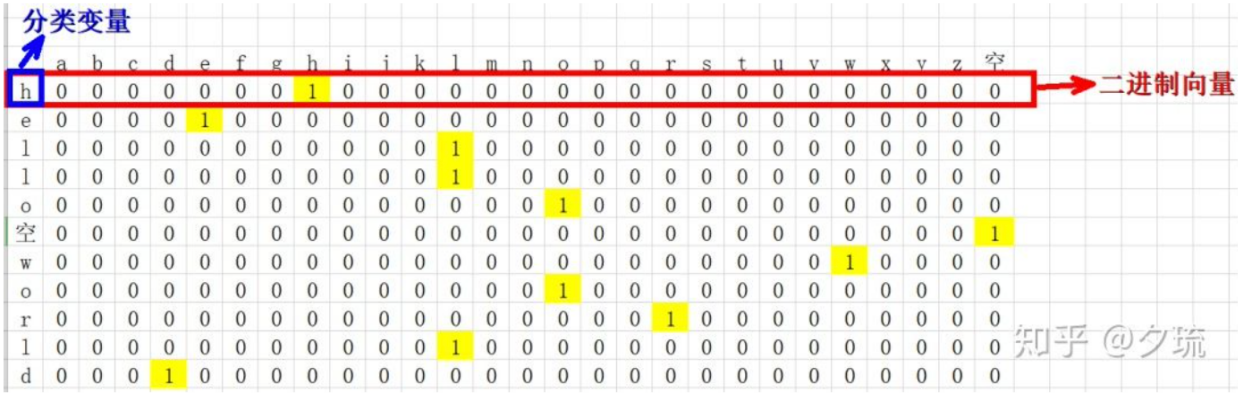
1) 将输入门、遗忘门变成两个门：更新门和重置门 重置门决定以往先前信息的程度

2) 将单元状态和输出合并为一个状态：H

(4). Embedding（词嵌入）

首先给每个字符一个对应下标，将每个单词对应数字表示，然后定义词嵌入，将每个单词转换为矩阵，矩阵中的元素可以不断更新，网络在训练过程更新词嵌入的参数，在计算通过 Var 访问

(5). One_hot（独热编码）



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	空
h	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
空	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

one-hot 编码是 N 位状态寄存器为 N 个状态进行编码的方式

使用 one-hot 编码中，我们可以将离散特征的取值扩展到欧式空间，在机器学习中，我们的研究范围就是在欧式空间中 Onehot 编码是分类变量作为二进制向量的表示。

这首先要求将分类值映射到整数值。然后，每个整数值被表示为二进制向量，除了整数的索引之外，它都是零值，它被标记为 1。

假设词典中不同字符的数量为 N(即词典大小 vocab_size),每个字符已经同一个从 0 到 N-1 的连续整数值索引一一对应。如果一个字符的索引是整数 i，那么我们创建一个全 0 的长为 N 的向量，并将其位置为 i 的元素设成 1。该向量就是对原字符的 one-hot 向量。

(6). perplexity （困惑度）

困惑度是对交叉熵损失函数做指数运算后得到的值。

- 最佳情况下，模型总是把标签类别的概率预测为 1，此时困惑度为 1；
- 最坏情况下，模型总是把标签类别的概率预测为 0，此时困惑度为正无穷；
- 基线情况下，模型总是预测所有类别的概率都相同，此时困惑度为类别个数。

程序实现

(1). 数据处理

```
'''
字符索引转换器：
对语料库中文本删去无用字符，将其用空格代替
set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据
删除重复字符，保证每个字符只出现一次
数字字符转换为文本，文本转换为向量
'''
class TextConverter(object):
    def __init__(self, corpus, max_vocab=5000):
        """
        建立一个字符索引转换器
            Args:
                corpus: 语料库
```

```
            max_vocab: 最大的单词数量
        """
        corpus = corpus.replace('\n', ' ').replace('\r', '
').replace(', ', ' ').replace('。', ' ').replace('《', ' ').replace('》
', ' ')
        # 去掉重复字符
        vocab = set(corpus)

        # 如果单词总数超过最大数值，去掉频率最低的
        vocab_count = {}

        # 计算单词出现频率并排序
        for word in vocab:
            vocab_count[word] = 0
        for word in corpus:
            vocab_count[word] += 1
        vocab_count_list = []
        for word in vocab_count:
            vocab_count_list.append((word, vocab_count[word]))
        vocab_count_list.sort(key=lambda x: x[1], reverse=True)

        # 如果超过最大值，截取频率最低的字符
        if len(vocab_count_list) > max_vocab:
            vocab_count_list = vocab_count_list[:max_vocab]
        vocab = [x[0] for x in vocab_count_list]
        self.vocab = vocab
        self.word_to_int_table = {c: i for i, c in enumerate(self.vocab)}
        self.int_to_word_table = dict(enumerate(self.vocab))

    def vocab_size(self):
        return len(self.vocab) + 1

    def word_to_int(self, word):
        if word in self.word_to_int_table:
            return self.word_to_int_table[word]
        else:
            return len(self.vocab)

    def int_to_word(self, index):
        if index == len(self.vocab):
            return '<UNK>'
        elif index < len(self.vocab):
```

```
            return self.int_to_word_table[index]
        else:
            return Exception('Unknown index!')

    def text_to_arr(self, text):
        arr = []
        for word in text:
            arr.append(self.word_to_int(word))
        return np.array(arr)

    def arr_to_text(self, arr):
        words = []
        for index in arr:
            words.append(self.int_to_word(index))
        return "".join(words)
```

(2). 读取数据库

```
# 加载周杰伦歌词数据集

def load_data_jay_lyrics():
    with zipfile.ZipFile('jaychou_lyrics.txt.zip') as zin:
        with zin.open('jaychou_lyrics.txt') as f:
            corpus_chars = f.read().decode('utf-8')
    # print(corpus_chars[:40])

    corpus_chars = corpus_chars.replace('\n', ' ').replace('\r', ' ')
    corpus_chars = corpus_chars[0:10000]
    idx_to_char = list(set(corpus_chars))
    char_to_idx = dict([(char, i) for i, char in enumerate(idx_to_char)])
    vocab_size = len(char_to_idx)
    corpus_indices = [char_to_idx[char] for char in corpus_chars]
    return corpus_indices, char_to_idx, idx_to_char, vocab_size
```

(3). 词向量提取方法

1) Embedding：

```
if self.fea_type == 'embed':
    self.word_to_vec = nn.Embedding(num_classes, embed_dim)
```

2) One_hot：

```
def one_hot(self, x, n_class, dtype=torch.float32):
    # X shape: (batch), output shape: (batch, n_class)
    x = x.long()
    res = torch.zeros(x.shape[0], n_class, dtype=dtype,
device=x.device)
    res.scatter_(1, x.view(-1, 1), 1)
    return res

def to_onehot(self, X):
    # X shape: (batch, seq_len), output: seq_len elements of (batch,
```

```
n_class)
    return [self.one_hot(X[:, i], self.num_classes) for i in
range(X.shape[1])]

    # 把输入的序列当成特征表示
```

(4). 建立网络

```
class CharRNN(nn.Module):
    def __init__(self, num_classes, embed_dim, hidden_size, num_layers,
dropout, device, fea_type='embed', rnn_type='RNN'):
        super(CharRNN, self).__init__()
        self.num_layers = num_layers
        self.hidden_size = hidden_size #hidden_size
        self.num_classes = num_classes
        self.fea_type = fea_type
        if self.fea_type == 'embed':
            self.word_to_vec = nn.Embedding(num_classes, embed_dim)
        else:
            self.word_to_vec = self.to_onehot

        # 主要看这 一部分 层数

        if rnn_type == 'RNN':
            self.rnn = nn.RNN(num_classes, hidden_size, num_layers)
        elif rnn_type == 'LSTM':
            self.rnn = nn.LSTM(num_classes, hidden_size, num_layers,
dropout)
        else:
            self.rnn = nn.GRU(num_classes, hidden_size, num_layers,
dropout)

        # 送到全连接层里：输入的神经元是隐层的 size 输出的神经元 一共的位数

        self.project = nn.Linear(hidden_size, num_classes)
        self.device = device
def forward(self, x, hs=None):
    # pdb.set_trace()
    batch = x.shape[0]
    if hs is None:
        hs = torch.zeros(self.num_layers, batch,
self.hidden_size).to(self.device)
    # pdb.set_trace()
    # word_embed = nn.functional.one_hot(x,
num_classes=self.num_classes)
    if self.fea_type == 'embed':
        word_embed = self.word_to_vec(x)   # (batch, len, embed) [128,
20, 512]
        word_embed = word_embed.permute(1, 0, 2)  # (len, batch, embed)
[20, 128, 512]
        out, h0 = self.rnn(word_embed, hs)
```

```
    else:
        word_embed = self.to_onehot(x)
        out, h0 = self.rnn(torch.stack(word_embed), hs)  # out: [20, 128, 512]  h0: [2, 128, 512]
        # word_embed = self.word_to_vec(x)  # (batch, len, embed) [128, 20, 512]
    # word_embed = word_embed.permute(1, 0, 2)  # (len, batch, embed) [20, 128, 512]
    # out, h0 = self.rnn(torch.stack(word_embed), hs)  # out: [20, 128, 512]  h0: [2, 128, 512]
    le, mb, hd = out.shape
    out = out.view(le * mb, hd)  # [2560, 512]
    out = self.project(out)  # [2560, 2581]
    out = out.view(le, mb, -1)  # [20, 128, 2581]
    out = out.permute(1, 0, 2).contiguous()  # (batch, len, hidden) [128, 20, 2581]

    # 输出出去 out

    return out.view(-1, out.shape[2]), h0  # [2560, 2581], [2, 128, 512]
```

(5). 定义预测函数

```
def predict_rnn(prefix, num_chars, rnn, params, init_rnn_state,
                num_hiddens, vocab_size, device, idx_to_char,
char_to_idx):
    state = init_rnn_state(1, num_hiddens, device)
    output = [char_to_idx[prefix[0]]]
    for t in range(num_chars + len(prefix) - 1):

        # 将上一时间步的输出作为当前时间步的输入

        X = to_onehot(torch.tensor([[output[-1]]], device=device),
vocab_size)

        # 计算输出和更新隐藏状态

        Y, state = rnn(X, state, params)
        if t < len(prefix) - 1:
            output.append(char_to_idx[prefix[t + 1]])
        else:
            output.append(int(Y[0].argmax(dim=1).item()))
        # pdb.set_trace()
    return ''.join([idx_to_char[i] for i in output])
```

(6). 梯度裁剪

当训练循环神经网络时，为了应对梯度爆炸，可以裁剪梯度。

```
def grad_clipping(params, theta, device):
    norm = torch.tensor([0.0], device=device)
    for param in params:
        norm += (param.grad.data ** 2).sum()
    norm = norm.sqrt().item()
```

```
    if norm > theta:
        for param in params:
            param.grad.data *= (theta / norm)
```

(7). 训练预测网络

```
def pick_top_n(preds, top_n=5):
    # 从概率分布，选取预测概率最大的前 5 个 得到可能性和标签
    top_pred_prob, top_pred_label = torch.topk(preds, top_n, 1)
    # 预测可能性归一化
    top_pred_prob /= torch.sum(top_pred_prob)
    # 将概率可能性转化成数组形式
    top_pred_prob = top_pred_prob.squeeze(0).cpu().numpy()
    # 将预测标签转化成数组形式
    top_pred_label = top_pred_label.squeeze(0).cpu().numpy()
    # 随机选取可能性最大五个标签中一个
    c = np.random.choice(top_pred_label, size=1, p=top_pred_prob)
    # 返回标签
    return c
"""
```

建立模型

模型可以定义成非常简单的三层，第一层是词嵌入，第二层是 RNN 层，

因为最后是一个分类问题，所以第三层是线性层，最后输出预测的字符。

```
"""
batch_size = 128
epochs = 200
num_classes = convert.vocab_size
embed_dim = num_classes
hidden_size = 512
num_layers = 2
dropout = 0.5
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
fea_type = 'embed'  # 'embed' or 'one_hot'
rnn_type = 'GRU'  # 'RNN' or 'LSTM' or 'GRU'
train_data = DataLoader(train_set, batch_size, shuffle=True)
model = CharRNN(num_classes, embed_dim, hidden_size, num_layers,
dropout, device, fea_type='onehot', rnn_type='RNN').to(device)
model = model.train()

criterion = nn.CrossEntropyLoss()       # 交叉熵损失函数

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

```
losses = []
for e in range(epochs):
    train_loss = 0
    for data in train_data:
        x, y = data
        # pdb.set_trace()
        x = x.long().to(device)  # 128 * 20
        y = y.long().to(device)
        # x = x.to(device)
        # y = y.to(device)

        # pdb.set_trace()
        score, _ = model(x)
        loss = criterion(score, y.view(-1))

        optimizer.zero_grad()
        loss.backward()
        # clip gradient
        nn.utils.clip_grad_norm_(model.parameters(), 1e-2)
        optimizer.step()

        train_loss += loss.item()
    print('epoch: {}, loss is: {:.3f}, perplexity is:
{:.3f}'.format(e+1, train_loss, np.exp(train_loss/len(train_data))))
    losses.append(train_loss)

    """
    生成文本
    给定了开始的字符，然后不断向后生成字符，将生成的字符作为新的输入再传入网络。
    这里需要注意的是，为了增加更多的随机性，在预测的概率最高的前五个里面依据他们的概率来进行随机选择。
    """
    # pdb.set_trace()
    with torch.no_grad():

        begin = '我睁开双眼望着空白'

        text_len = 30
        model = model.eval()
        samples = [convert.word_to_int(c) for c in begin]
        input_txt = torch.LongTensor(samples)[None].to(device)
        _, init_state = model(input_txt)
        result = samples
        model_input = input_txt[:, -1][:, None]
        for i in range(text_len):
            out, init_state = model(model_input, init_state)
            # pdb.set_trace()
            pred = pick_top_n(out, top_n=5)
            # pred = out.argmax(dim=1).item()
```

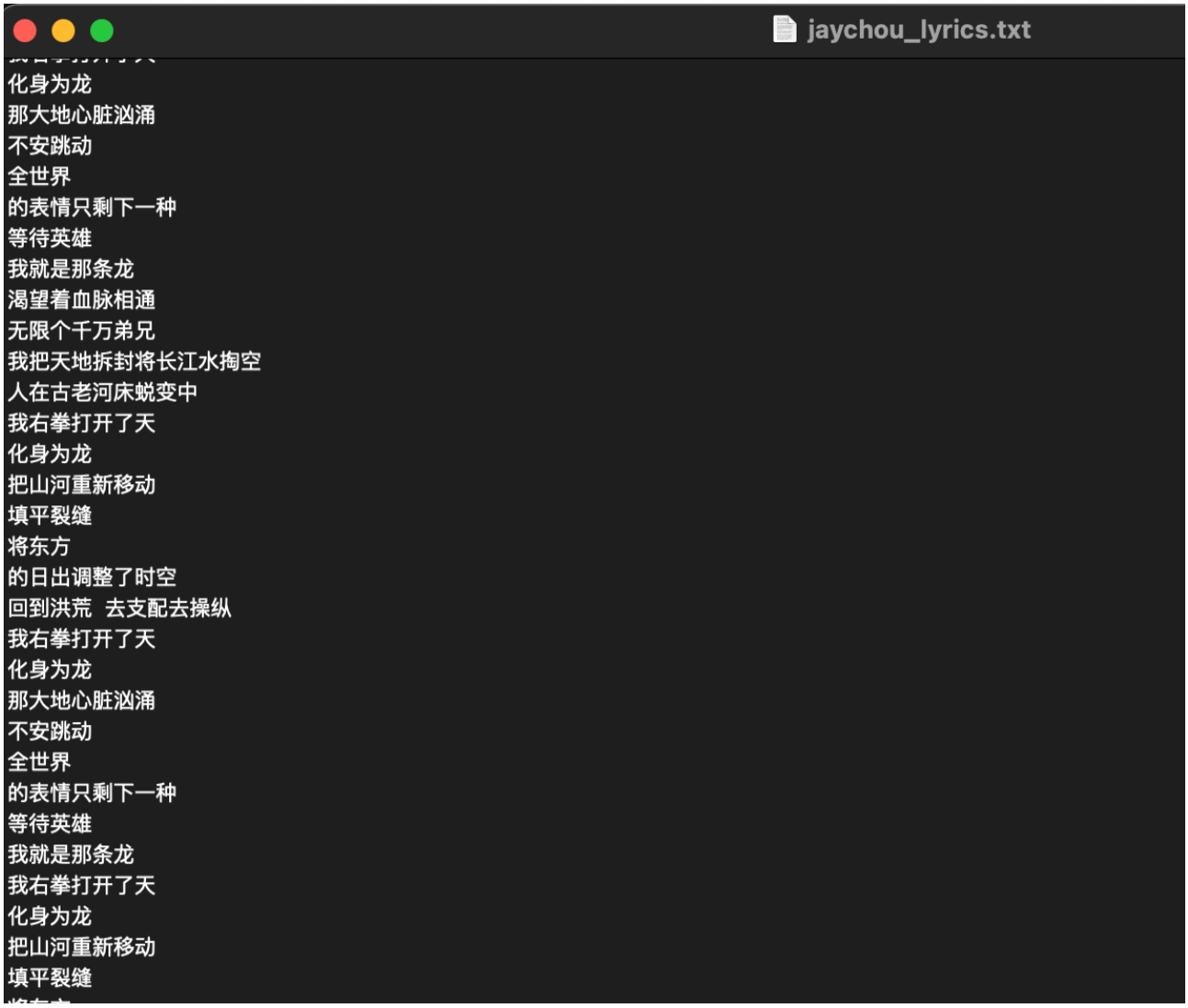
```
            result.append(pred[0])
        text = convert.arr_to_text(result)
        text = text.replace('<UNK>', ' ')
        print('Generate text is: {}'.format(text))
    model.train()
```

(6). 显示训练结果

```
plt.figure()
plt.plot(losses)
plt.xlabel('epoches')
plt.ylabel('losses')
x = range(0, 210, 20)
plt.xticks(x)
y = range(0, 210, 20)
plt.yticks(y)
plt.title('GRU_one_hot_losses')
plt.savefig('GRU_one_hot:losses.png')
plt.show()
```

3. 实验

3.1 数据库介绍



收集了周杰伦从第一张专辑《Jay》到第十张专辑《跨时代》中的歌词

3.2 定量实验结果展示

```
RNN_embed:
epoch: 100, loss is: 4.982, perplexity is: 1.221
Generate text is: 我睁开双眼看着空白发发发发鸽鸽发鸽发 杨鸽玫 鸽鸽 分分发发玫  鸽玫 玫分发
epoch: 200, loss is: 4.444, perplexity is: 1.195
Generate text is: 我睁开双眼看着空白玫玫玫慢你玫玫色意超发玫发发 来玫 来  来来原原原 发玫原
epoch: 300, loss is: 4.318, perplexity is: 1.189
Generate text is: 我睁开双眼看着空白 杨i 发发泣倒泣花发发发调杨发a发调调发泣发调间泣间 调片
```

```
RNN_on_hot:
epoch: 100, loss is: 4.402, perplexity is: 1.193
Generate text is: 我睁开双眼看着空白 的衅  落了 杨之进边了春了  咸的舞的着帮的了 发的白腐
epoch: 200, loss is: 3.968, perplexity is: 1.172
Generate text is: 我睁开双眼看着空白两活睡进 海壁移  默春 蔓  得着进玫  睡掉事明 箱誉活明
epoch: 300, loss is: 3.828, perplexity is: 1.165
Generate text is: 我睁开双眼看着空白写风后  草景方的 霍草入敲敲活响默敲色花入默的敲敲敲敲 色
```

```
LSTM_on_hot:
epoch: 100, loss is: 69.121, perplexity is: 15.876
Generate text is: 我睁开双眼看着空白白  的得 都的的得着的着 得得  了得了了得  得  的的
epoch: 200, loss is: 20.108, perplexity is: 2.235
Generate text is: 我睁开双眼看着空白里 色的的了得 的得了 得的好 的得得说 好 的 了的泪里
epoch: 300, loss is: 6.544, perplexity is: 1.299
Generate text is: 我睁开双眼看着空白落着始 的的得的  着的着好看好好看去的 了的的了 了
```

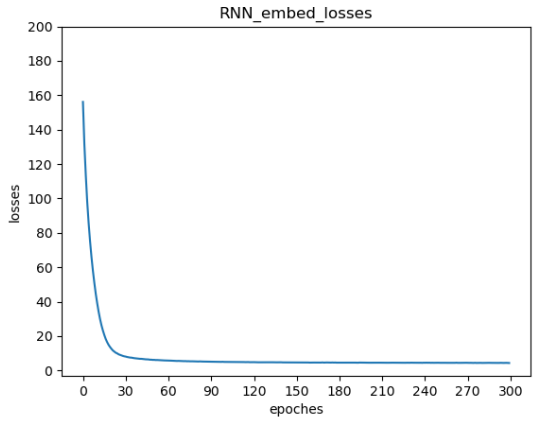
```
LSTM_embed:
epoch: 100, loss is: 9.095, perplexity is: 1.439
Generate text is: 我睁开双眼看着空白 得道得 诺 得帮  得  有 有有有  有 听诺有有有得诺
epoch: 200, loss is: 6.130, perplexity is: 1.278
Generate text is: 我睁开双眼看着空白一忘入  有刻 入入  入机入入入机机机刻 入 发 入
epoch: 300, loss is: 5.520, perplexity is: 1.247
Generate text is: 我睁开双眼看着空白  油  听 油玫穿  玫忘玫油玫  忘穿 穿穿玫油 玫
```

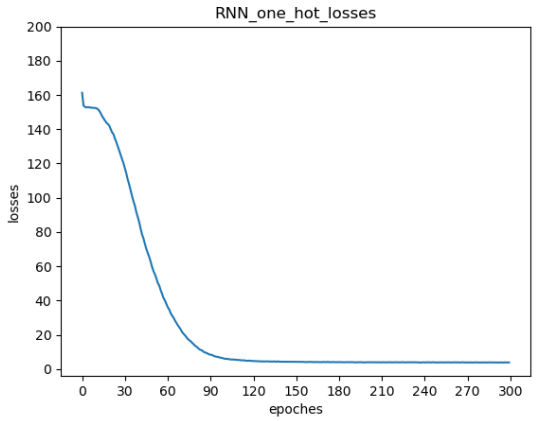
```
GRU_embed:
epoch: 100, loss is: 8.069, perplexity is: 1.381
Generate text is: 我睁开双眼看着空白忘忘忘 停停我停记味 记我停 我我味记我 记停记停停  结
epoch: 200, loss is: 6.785, perplexity is: 1.312
Generate text is: 我睁开双眼看着空白  什间  玫玫间 玫 绕玫发绕发 绕 间 玫发  鸽玫玫鸽
epoch: 300, loss is: 6.476, perplexity is: 1.296
Generate text is: 我睁开双眼看着空白角鸽发鸽  玫 玫  玫趁玫鸽玫 鸽鸽角忘  玫忘玫 玫角
```

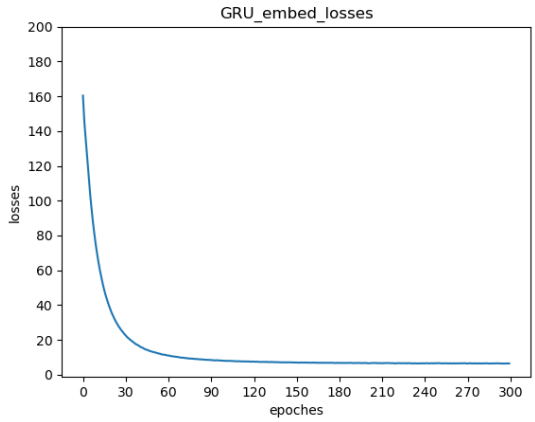
```
GRU_one_hot:
epoch: 100, loss is: 8.704, perplexity is: 1.416
Generate text is: 我睁开双眼看着空白话 婆发 誓发色婆  的 誓的誓誓 发发的 的泪的  默誓
epoch: 200, loss is: 4.305, perplexity is: 1.188
Generate text is: 我睁开双眼看着空白现 久  更模足蓝  娘移移  模  更娘娘模更 娘
epoch: 300, loss is: 3.794, perplexity is: 1.164
Generate text is: 我睁开双眼看着空白久发发的  玫错 频发 玫婆发发  早  婆频婆  发发婆频早
```

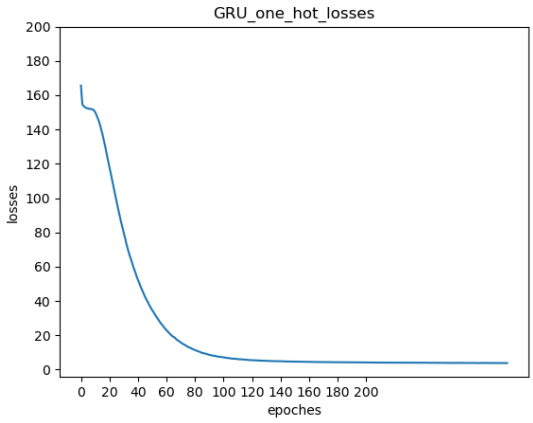
Model	epoch	loss	perplexity
RNN_embed	100	4.982	1.221
	200	4.444	1.195
	300	4.318	1.189
RNN_on_hot	100	4.402	1.193
	200	3.968	1.172
	300	3.828	1.165
LSTM_embed	100	9.095	1.439
	200	6.13	1.278
	300	5.52	1.247
LSTM_on_hot	100	69.121	15.876
	200	20.108	2.235
	300	6.544	1.299
GRU_embed	100	8.069	1.381
	200	6.785	1.312
	300	6.476	1.296
GRU_one_hot	100	8.704	1.416
	200	4.305	1.188
	300	3.794	1.164

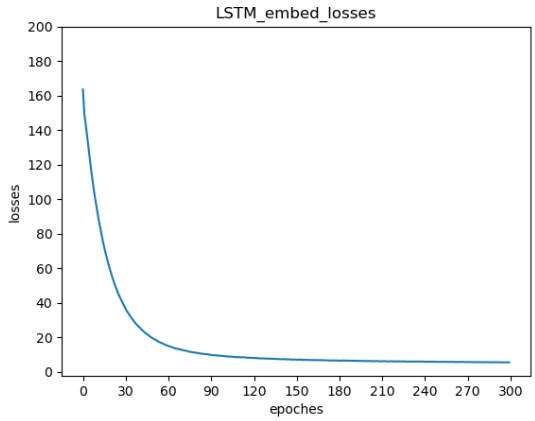
3.3 定性实验结果展示

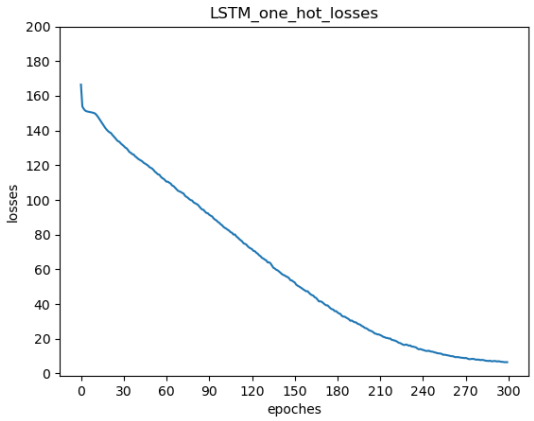












3.4 分析实验结果

Model	epoch	loss	perplexity
RNN_embed	300	4.318	1.189
RNN_on_hot	300	3.828	1.165
LSTM_embed	300	5.52	1.247
LSTM_on_hot	300	6.544	1.299
GRU_embed	300	6.476	1.296
GRU_one_hot	300	3.794	1.164

1.由实验数据可知，在相同网络模型情况下，训练三百次对歌词进行预测使用不同语言模型对比 embedding 与 one_hot 模型的 loss 和 perplexity 可知：使用 RNN 与 GRU 网络下，one_hot 效果较 embedding 好，这是因为 one_hot 编码能分类变量作为二进制向量的表示，每个样本只对应于一个类别（即只在对应的特征处值为 1，其余地方值为 0），而我们的分类结果，得到是隶属于某个类别的概率，这样在进行损失函数或准确率计算时，变得非常方便。

2.由实验数据可知，在相同语言模型情况下，训练三百次网络对歌词进行预测 RNN 和 GRU 效果差异不大且较 LSTM 好，loss 与困惑度较低，其中 GRU_one_hot 效果最佳

3.GRU 的输入输出结构与普通的 RNN 是一样的。

有一个当前的输入 xt ，和上一个节点传递下来的隐状态（hidden state），这个隐状态包含了之前节点的相关信息。

结合 xt 和 ht-1，GRU 会得到当前隐藏节点的输出 yt 和传递给下一个节点的隐状态 ht 。使用一个门控同时可以进行遗忘和选择，记忆参数比 LSTM 少，训练成本更少，但是却也能够达到与 LSTM 相当甚至更佳的功能。