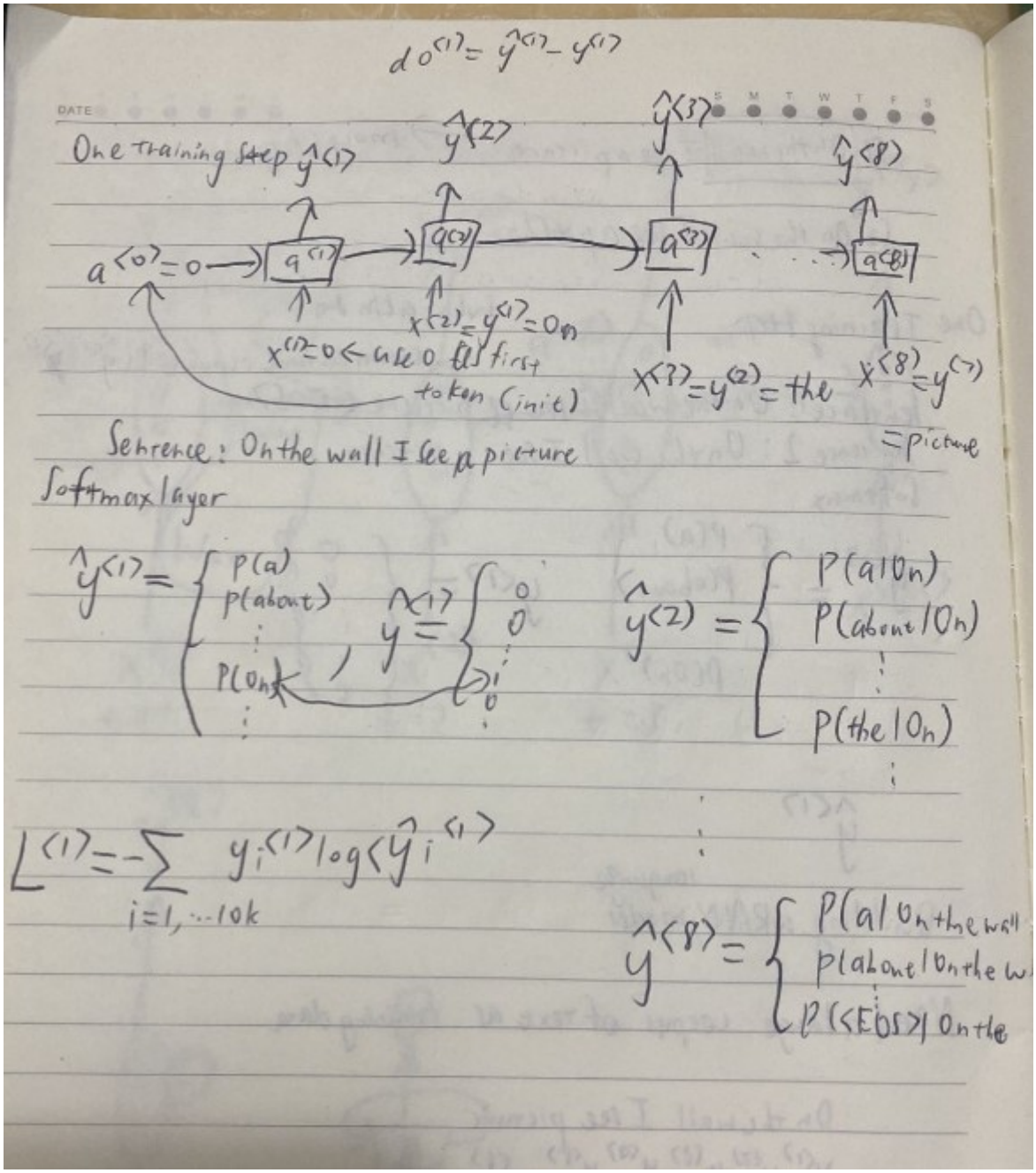
期末实验

陈浚铭（20337021）

通过这次实验，我深入理解到 rnn 神经网络在语言模型的原理。基于 RNN 的语言模型分为训练，评估和测试三个过程。在训练过程里， 我们会不断的更新 RNN 模型的权重, 使得收敛到一个好的模型。RNN 模型如下图所显示：



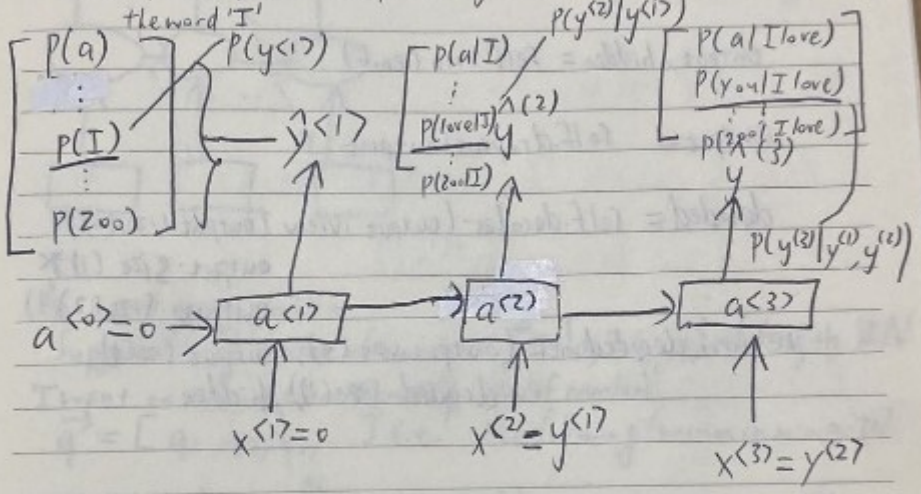
从图上可见， 我们的输入是 x 输出 y,而而中间就有 hidden layer(隐藏层）。因此，x 经过隐藏层的学习过程从而输出到 y。但是， 我们注意到从左到右的方向隐藏层的学习的过程。从左到右，右面的 x 是会通过前面左面的 x 来改变自己的值。因此，隐藏层学习和更新权重的方向是向下到上，向左到右的。之后，在预句子方面， 我们需要计算句子出现的概率。对于一个句子 $x = x_1x_2...x_n$， 我们计算句子的概率为 $P(x_1 ,x_2,…,x_n) = P(x1)P(x2|x1)…P(x_n|x_1 ,x_2,…,x_{n-1})$。这个概率等式是通过 chain rule of probability 的出来的。以下图为一个例子：

Compute sentence probability

Given trained model, and a new sentence

$(y^{<1>}, y^{<2>}, y^{<3>}) = (I, love, you)$

The first prediction is bunch a probability for all vocabs, in which we can find the word 'I'



Sentence Probability

$$P(\text{sentence}) = P(y^{(1)})\, P(y^{(2)}|y^{(1)})\, P(y^{(3)}|y^{(1)}, y^{(2)})$$

The best language model should give the highest probability to unseen test data

实验代码：

首先我是用到 nltk 的库来对于训练，评估和预测的文本分句子(sentence_tokenize 函数)以及对于句子分词(word_tokenize 函数)。

```
with open("train_en.txt", "r") as file:
    content = file.read()
sentences = sent_tokenize(content)
for sent in sentences:
    split_sent = word_tokenize(sent)
    lines.append(split_sent)
    for word in split_sent:
        words.add(word)

with open("eval_en.txt", "r") as file:
    eval_content = file.read()
e_lines = []
e_sentences = sent_tokenize(eval_content)
for sentence in e_sentences:
    e_split_sentence = word_tokenize(sentence)
    e_lines.append(e_split_sentence)
    for word in e_split_sentence:
        words.add(word)

with open("test_en.txt", "r") as file:
    test_content = file.read()
test_lines = []
test_sentences = sent_tokenize(test_content)

for sentence in test_sentences:
    test_split_sentence = word_tokenize(sentence)
    test_lines.append(test_split_sentence)
    for word in test_split_sentence:
        words.add(word)
```

把它们加到 dataloader:

```
news_ds = Article_Dataset(word2id, id2word, lines)
train_dataloader = DataLoader(news_ds)


eval_ds = Article_Dataset(word2id, id2word, e_lines)
eval_dataloader = DataLoader(eval_ds)

test_ds = Article_Dataset(word2id, id2word, test_lines)
test_dataloader = DataLoader(test_ds)
```

之后，基于示例代码的 rnn 模型，我加上了预测函数 predict()

```
def predict(model, dataloader, data_lines):
    model.eval()
    with torch.no_grad():
        hidden = model.init_hidden(BATCH_SIZE, requires_grad = False)
        for i, batch in enumerate(dataloader):
            data, target = batch

            hidden = repackage_hidden(hidden)
            with torch.no_grad():
                output, hidden = model(data, hidden)
            sentence_probability = 1
            prob = nnf.softmax(output, dim=1)
            for word in data_lines[i]:
                print(word, end='')
                sentence_probability *= prob.numpy()[0][i][word2id[word]]
            print(data_lines[i], end= '')
            print(sentence_probability)
```

在这里，我通过前面所说的计算句子概率的公式写。注意到我们在 test_en.txt 有两个句子而已。
注意到我们的 output 这个 tensor 并不是概率的 tensor, 我们需要通过 softmax 函数把它转变为概率，然后，我们对于句子的每一个字， 计算它对应的概率或者条件概率，计算它们的乘积就可以得到句子概率。

实验结果：

```
Best model, epoch:  39 loss:  0.6305528879165649
epoch  44  iteration:  295 loss =  0.17916512489318848
epoch  49  iteration:  295 loss =  0.8285380005836487
epoch  54  iteration:  295 loss =  0.535076379776001
epoch  59  iteration:  295 loss =  0.44880762696266174
Best model, epoch:  59 loss:  0.44880762696266174
epoch  64  iteration:  295 loss =  0.1410582959651947
epoch  69  iteration:  295 loss =  0.229578897356987
Best model, epoch:  69 loss:  0.229578897356987
epoch  74  iteration:  295 loss =  0.22896640002727509
epoch  79  iteration:  295 loss =  0.47832345962524414
epoch  84  iteration:  295 loss =  0.47827863693237305
epoch  89  iteration:  295 loss =  0.6426665186882019
epoch  94  iteration:  295 loss =  0.2904573380947113
epoch  99  iteration:  295 loss =  0.5580669045448303
Idon't knowwhyIlikethisIdon't knowwhyIlikethismoviesoThisisagoodfilm.Sentence:  I don't know why I like this
I don't know why I like this movie so
This is a good film. probability =  1.962619048977905e-119
ThisisverySentence:  This is very probability =  8.337254094630201e-05
```

P(“I don't know why I like this I don't know why I like this movie so this is a good film”)
= 1.962619048977905e-119
P(“This is very”) = 8.337254094630201e-05
当中注意到第一个句子的概率比第二个句子的概率小, 这是因为第一个句子比较长，所以得到的概率会比较小。

我们发分别知道 RNN 语言模型的优点/缺点为：
RNN 语言模型优点：
1、可以处理任意长度的输入，长的输入不会增加模型的规模；
2、由于每个时间步需要考虑前一个时间步的计算激活值，每一个时间步的计算可以利用多个时间步之前的结果；
3、每个时间步的权重矩阵都是共享的，故学习结果也是可以共享的，学习效率高；
RNN 语言模型优点：
1、RNN 计算慢，因为每一个时间步都需要前一个时间步的计算结果，不同的输入不能并行处理，只能一个接一个的来；
2、在实践中仍然较难顾及到多个时间步之前的信息。

实验心得

通过这次实验，我更深入理解到基于 RNN 的语言模型的实验，而且能够更深入理解 pytorch 库的功能操作等。