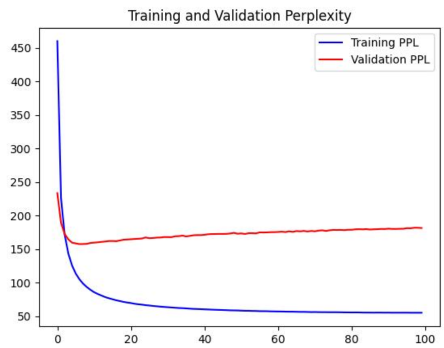
Part1:RNN

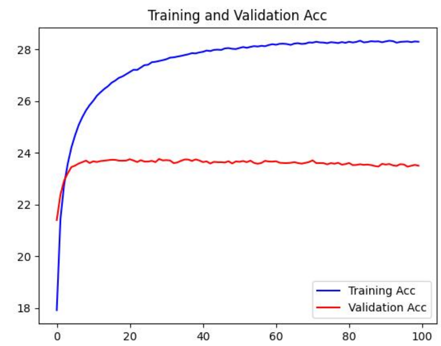
1.1:

- 训练脚本：main_rnn_basic.py
- 使用模型：model/rnn.py

训练与测试结果：

- Best Validation PPL：157.68934232462753
- Curves are as follows





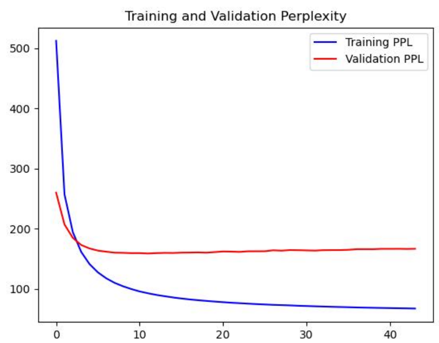
-

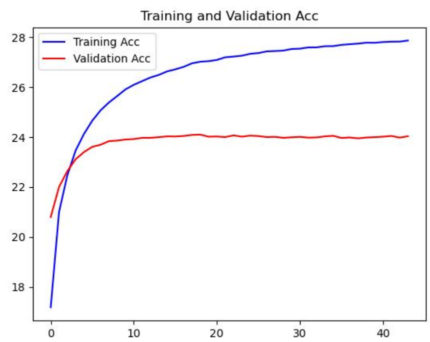
1.2:

- 训练脚本：main_rnn_basic.py
- 使用模型：model/LSTM.py

训练与测试结果：

- Best Validation PPL：159.7654981745692
- Curves are as follows





-

1.3

- Input:prompt = ["The", "player", "progress", "through", "a", "serious", "of"]

```
size of train set:  2088628
size of valid set:  217646
device: cuda:0
device: cuda:0
The player progress through a serious of the game .
```

- 训练代码如下：

```
def predict(my_net):
prompt = ["The", "player", "progress", "through", "a",
"serious", "of"]
for m in prompt:
print(m, end=' ')
voc = []
for word in prompt:
voc.append(data_loader.word_id[word])
prompt = torch.LongTensor(voc).unsqueeze(-1).to(device)
my_net.eval()
output,hidden = my_net(prompt)
for i in range(20):
last = output[-1, -1]
idx = torch.argmax(last)
if data_loader.vocabulary[idx]== '<eos>':
break
voc.append(idx)
print(data_loader.vocabulary[idx],end=' ')
prompt = torch.LongTensor(voc).unsqueeze(-1).to(device)
#将新生成的字符加到promt尾部，使其成为一个新的输入。
output, hidden = my_net(prompt, hidden)

print()
```

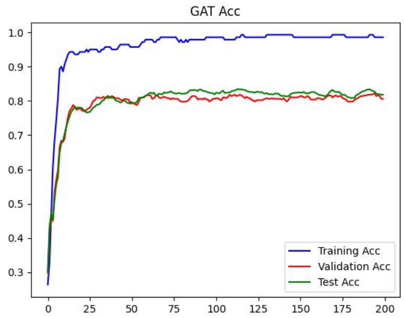
1.4:

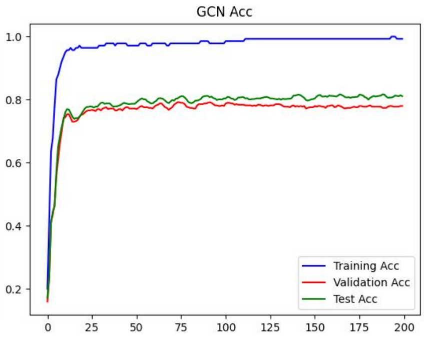
- Char-based LM 的优势：Char-based LM 的字典大小要远远小于 Word-based LM，而且在有未知单词的情况也可以预测。
- Char-based LM 的劣势：同样的一个句子，在 Char-based 中转换成 Tokens 序列更长，因此在训练过程中必须考虑时间方向上更多的标记之间的依赖关系，需要更大的隐藏层才能刻画。同时，Char-based LM 相对于 Word-based LM 还需要学习拼写。

Part2:GNN and Node2Vec:

2.1:

其中的 GAT 与 GCN 的表现如下，其都具有较好的可训练性，对于给定的数据可以随着训练次数增加准确性提高。





2.2:

该模型表现良好，其准确性可以随着数据量的增多而稳定上升。