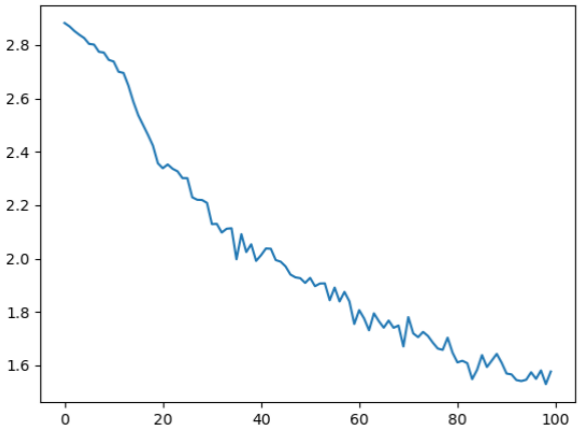
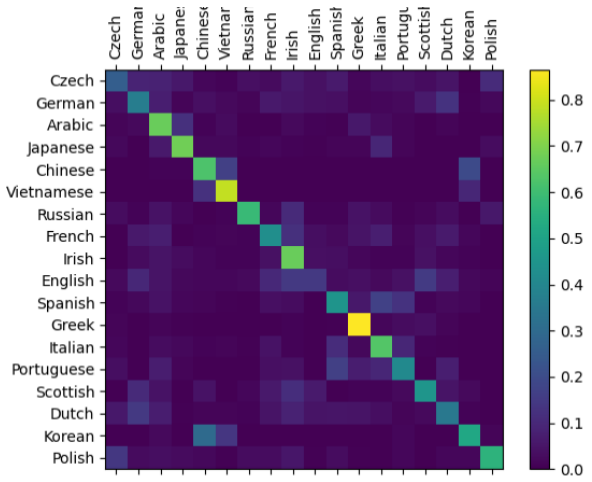
深度学习报告w4：循环神经网络

1 RNN

网络结构

```
RNN(
  (i2h): Linear(in_features=185, out_features=128, bias=True)
  (i2o): Linear(in_features=185, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

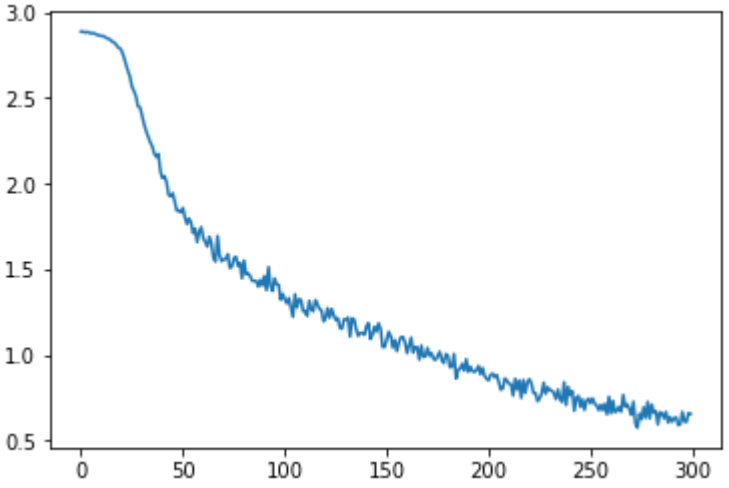


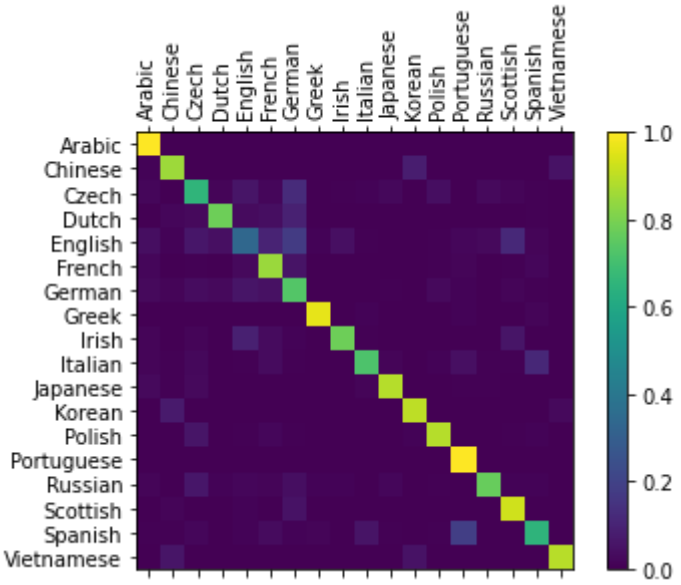


2 LSTM

网络结构

```
LSTM(
  (rnn): LSTM(57, 64)
  (out): Linear(in_features=64, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=-1)
)
```





3 LSTM的优势

1. 长期依赖问题

RNN:

- RNN处理序列数据时，每一步的输出依赖于前一步的输出。这使得RNN在处理短期依赖时表现良好，但在处理长期依赖时存在问题。原因在于，RNN在每一步更新时，梯度会通过时间反向传播（BPTT，Backpropagation Through Time），在长序列中，梯度会逐渐变小（梯度消失问题）或变大（梯度爆炸问题），这导致远距离信息难以被有效地传递和学习。

LSTM:

- LSTM通过引入记忆单元（Cell State）和门控机制（输入门、遗忘门和输出门）来有效地解决长期依赖问题。记忆单元可以直接传递信息，且在每一步通过门控机制选择性地更新或保留信息，从而减轻了梯度消失和爆炸问题。这样，LSTM能够更好地保留长时间跨度的信息。

2. 门控机制

RNN:

- 传统RNN缺乏门控机制，无法有效控制信息的流动和存储。它仅仅通过单一的递归层来处理输入信息和隐藏状态之间的关系，这使得它在处理复杂的序列依赖时表现不佳。

LSTM:

- LSTM的门控机制包括输入门、遗忘门和输出门，这些门能够对输入信息、记忆单元状态和输出信息进行选择性控制。输入门控制新信息的引入，遗忘门控制记忆单元状态的更新，输出门控制从记忆单元输出的信息。这些门的引入使得LSTM能够更灵活和精细地处理序列数据，从而提高了模型的表现。

3. 训练效率和稳定性

RNN：

- 由于梯度消失和爆炸问题，RNN在训练长序列时会遇到显著的困难。这不仅影响训练效率，还影响模型的稳定性和收敛性。模型可能会陷入局部最优，或者训练时间大幅增加。

LSTM：

- LSTM通过记忆单元和门控机制的设计，有效减轻了梯度消失和爆炸问题，使得训练更加稳定和高效。LSTM在长序列训练中能够更好地保持信息流的稳定性，从而加快训练速度和提高模型的收敛性。

4. 实际应用表现

在实际应用中，如自然语言处理、时间序列预测、语音识别等领域，LSTM通常表现出更好的准确性和鲁棒性。这是因为LSTM能够更好地捕捉长时间跨度的依赖关系，并且其复杂的门控机制允许它更灵活地处理输入数据。

4 自己实现LSTM

LSTM 单元的前向传播

LSTM单元由三个门（输入门、遗忘门和输出门）以及一个单元状态组成。每个门都有相应的权重和偏置。前向传播过程如下：

1. 计算遗忘门：决定单元状态中的哪些部分需要被遗忘。
2. 计算输入门：决定哪些新的信息将被存储到单元状态中。
3. 计算候选单元状态：生成新的候选单元状态。
4. 更新单元状态：结合遗忘门和输入门来更新单元状态。
5. 计算输出门：决定当前单元的输出。

```
class MyLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MyLSTM, self).__init__()
        self.hidden_size = hidden_size

        # LSTM parameters
        self.W_f = nn.Linear(input_size + hidden_size, hidden_size)
        self.W_i = nn.Linear(input_size + hidden_size, hidden_size)
        self.W_C = nn.Linear(input_size + hidden_size, hidden_size)
        self.W_o = nn.Linear(input_size + hidden_size, hidden_size)

        # Output layer
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=-1)
```

```
    def forward(self, input, h, C):
        combined = torch.cat((input, h), dim=2)

        f_t = torch.sigmoid(self.W_f(combined))
        i_t = torch.sigmoid(self.W_i(combined))
        C_tilde = torch.tanh(self.W_C(combined))
        C = f_t * C + i_t * C_tilde
        o_t = torch.sigmoid(self.W_o(combined))
        h = o_t * torch.tanh(C)

        output = self.out(h)
        output = self.softmax(output)
        return output, h, C

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size), torch.zeros(1, 1,
self.hidden_size)

n_hidden = 64
rnn = MyLSTM(n_letters, n_hidden, n_categories)
```