南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

深度学习与应用实验报告

RNN 循环神经网络

艾明旭　2111033

年级：2021 级

专业：信息安全

指导教师：侯淇彬

2024 年 4 月 22 日

目录

一、 基础 RNN 结构

RNN 称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。

(一) 基础 RNN 的网络结构

```
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)
)
```

基础的 RNN 网络为一个两层的网络，其映射关系如上面的结构所示。

(二) 图形展示实验结果

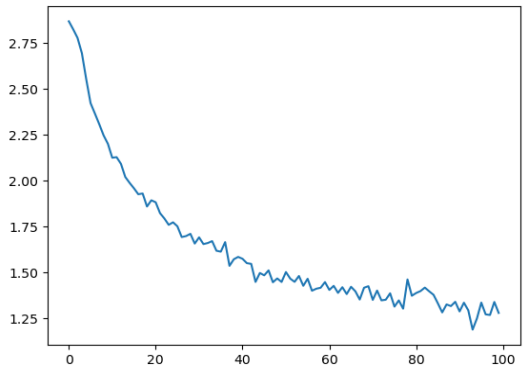

图 1: 基础 RNN validation loss

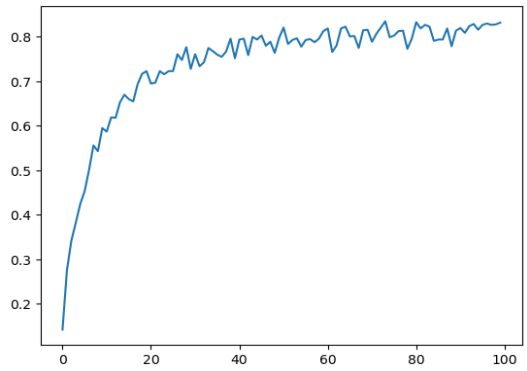

图 2: 基础 RNN validation accuracy

二、 LSTM

(一) LSTM 的实现

长短期记忆（Long short-term memory, LSTM）是一种特殊的 RNN，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。简单来说，就是相比普通的 RNN，LSTM 能够在更长的序列中有更好的表现。

LSTM 内部主要有三个阶段：

1. 忘记阶段。这个阶段主要是对上一个节点传进来的输入进行选择性忘记。简单来说就是会“忘记不重要的，记住重要的”。

具体来说是通过计算得到的 z^f （f 表示 forget）来作为忘记门控，来控制上一个状态的 c^{t-1} 哪些需要留哪些需要忘。

2. 选择记忆阶段。这个阶段将这个阶段的输入有选择性地进行“记忆”。主要是会对输入 x^t 进行选择记忆。哪些重要则着重记录下来，哪些不重要，则少记一些。当前的输入内容由前面计算得到的 z 表示。而选择的门控信号则是由 z^i （i 代表 information）来进行控制。

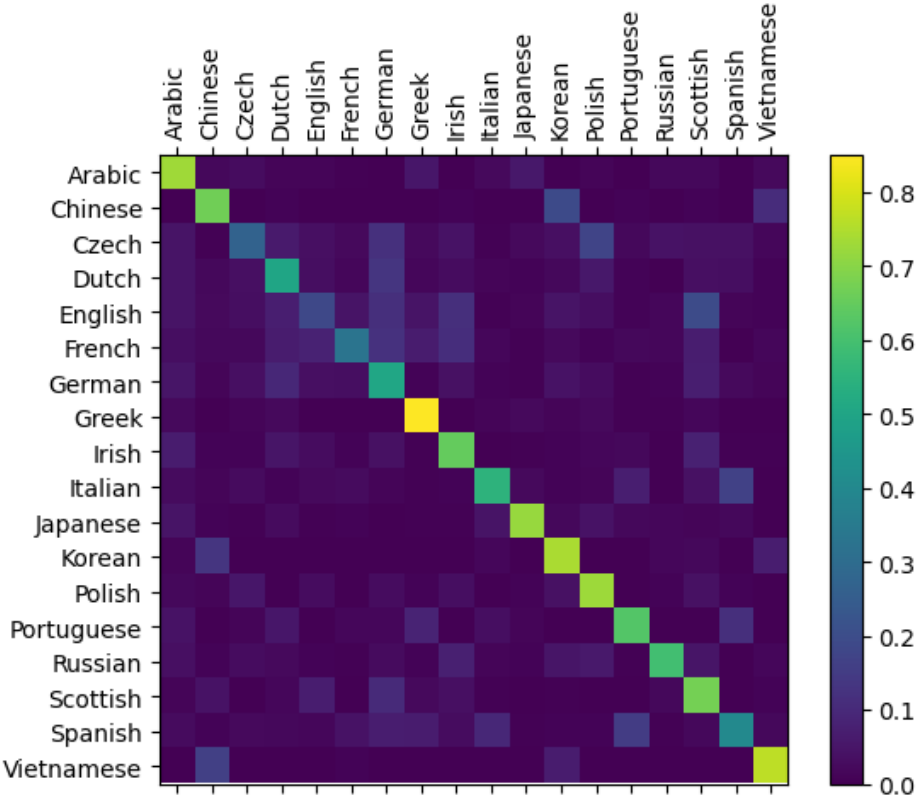

图 3: 原始 RNN 预测矩阵

3. 输出阶段。这个阶段将决定哪些将会被当成当前状态的输出。主要是通过 z^o 来进行控制的。并且还对上一阶段得到的 c^o 进行了放缩（通过一个 tanh 激活函数进行变化）。

与普通 RNN 类似，输出 y^t 往往最终也是通过 h^t 变化得到。

```
LSTM(
  (rnn): LSTM(57, 128)
  (out): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

(二) word2vec 基本原理

Word Embedding 是将「不可计算」「非结构化」的词转化为「可计算」「结构化」的向量。

CBOW 和 Skip-gram 是 Word2vec 的两种训练模式。CBOW 是通过上下文来预测当前值。相当于一句话中扣掉一个词，让你猜这个词是什么。Skip-gram 是用当前词来预测上下文。相当于给你一个词，让你猜前面和后面可能出现什么词。

为了提高速度, Word2vec 经常采用 2 种加速方式: Negative Sample 和 Hierarchical Softmax

优点：由于 Word2vec 会考虑上下文，跟之前的 Embedding 方法相比，效果要更好（但不如 18 年之后的方法）；比之前的 Embedding 方法维度更少，所以速度更快; 通用性很强，可以用在各种 NLP 任务中。

缺点：由于词和向量是一对一的关系，所以多义词的问题无法解决。Word2vec 是一种静态的方式，虽然通用性强，但是无法针对特定任务做动态优化

(三) 结果图像展示

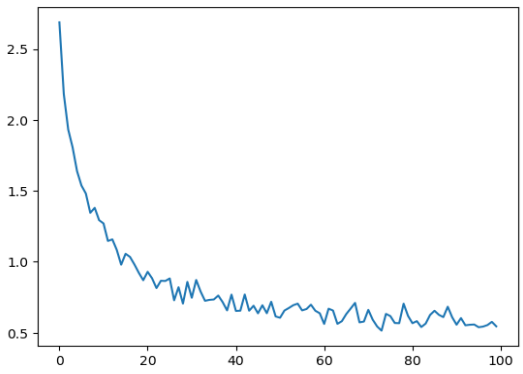

图 4: 基础 LSTM validation loss

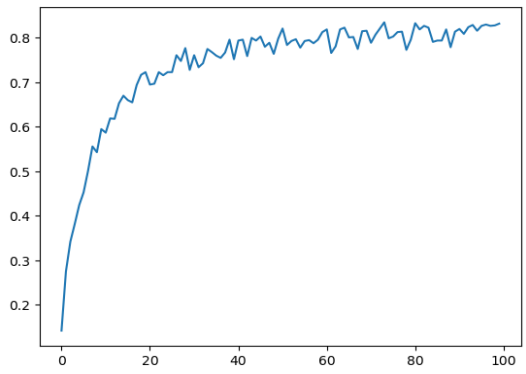

图 5: 基础 LSTM validation accuracy

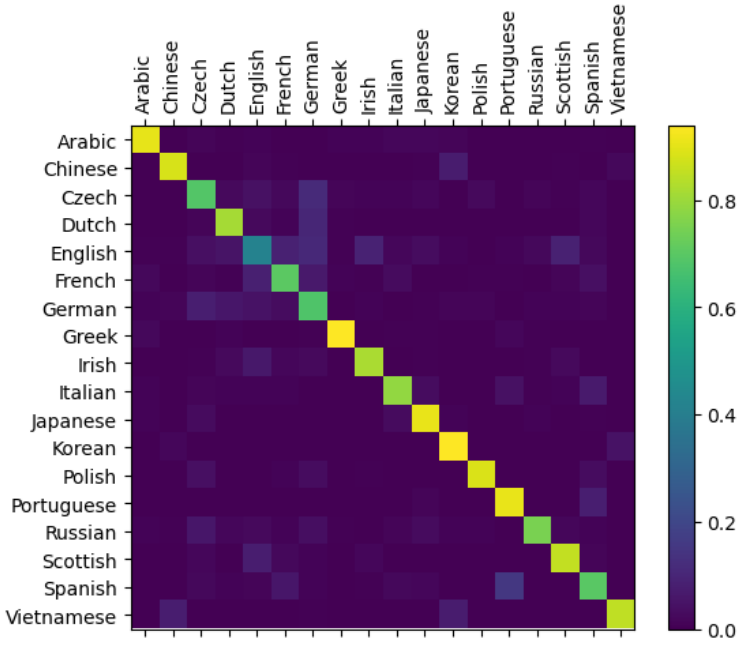

图 6: 原始 LSTM 预测矩阵

可以看到，在引入时间长短因子之后，整个深度学习的网络层次都发生了一定的变化。

三、 LSTM 与 RNN 的区别

LSTM（Long Short-Term Memory）网络优于 RNN（Recurrent Neural Network）网络的主要原因是它能够更好地处理长期依赖问题。

RNN 的隐藏状态更新公式为：$h_t = tanh(W_h h * h_{t-1} + W_x h * x_t + b_h)$ 其中，h_t 是当前时间步的隐藏状态，h_{t-1} 是前一时间步的隐藏状态，x_t 是当前时间步的输入，$W_h h$ $W_x h$ 和 b_h 是可学习的参数。

LSTM 的隐藏状态和单元状态更新公式为：$i_t = sigmoid(W_i i * x_t + b_i i + W_h i * h_{t-1} + b_h i) f_t = sigmoid(W_i f * x_t + b_i f + W_h f * h_{t-1} + b_h f) g_t = tanh(W_i g * x_t + b_i g + W_h g * h_{t-1} + b_h g) o_t =$

$sigmoid(W_io * x_t + b_io + W_ho * h_{t-1} + b_ho)c_t = f_t * c_{t-1} + i_t * g_t h_t = o_t * tanh(c_t)$

其中，i_t f_t g_t 和 o_t 分别是输入门、遗忘门、单元状态和输出门，c_t 是当前时间步的单元状态，h_t 是当前时间步的隐藏状态，W 和 b 是可学习的参数。

1. 长期依赖问题：在处理序列数据时，如果当前的输出与过去的输入有很长的时间跨度，那么 RNN 往往难以建立这种长期的联系。这是因为 RNN 在反向传播时，梯度往往会发生消失（值变得非常小）或爆炸（值变得非常大），导致网络难以学习到这种长期的依赖关系。而 LSTM 通过引入门控机制（包括输入门、遗忘门和输出门）和单元状态，能够更好地保持长期的信息，从而解决了这个问题。

2. 遗忘机制：LSTM 的遗忘门可以决定什么信息应该被遗忘，什么信息应该被保留。这使得 LSTM 能够在处理序列数据时，更好地区分重要和不重要的信息。

3. 更好的性能：在许多任务中，如语音识别、语言模型、机器翻译等，LSTM 都表现出了比 RNN 更好的性能。

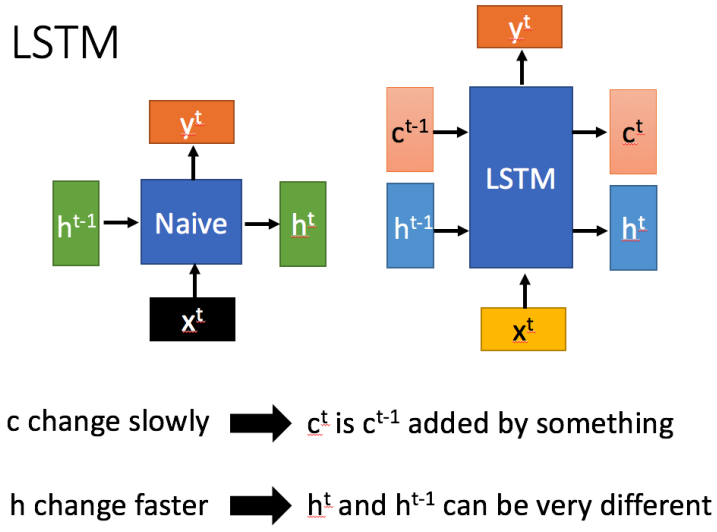

图 7: LSTM 与 RNN 的区别

四、 带有 Depthwise conv 的 MobileNets

(一) 自己编写的 LSTM

这里我自己定义了一个 LSTM 方法，并且定义了 init,init_weights,forward 三个函数用来展示图形。

```
class MyLSTM(nn.Module):
    def __init__(self, input_sz, hidden_sz,output_sz):
        super().__init__()
        self.input_sz = input_sz
        self.hidden_size = hidden_sz
        self.W = nn.Parameter(torch.Tensor(input_sz, hidden_sz * 4))
        self.U = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz * 4))
        self.bias = nn.Parameter(torch.Tensor(hidden_sz * 4))
```

```
        self.init_weights()
    def init_weights(self):
        stdv = 1.0 / math.sqrt(self.hidden_size)
        for weight in self.parameters():
            weight.data.uniform_(-stdv, stdv)
    def forward(self, x, y, z, init_states=None):
        """Assumes x is of shape (batch, sequence, feature)"""
        bs, seq_sz, _ = x.size()
        hidden_seq = []
        if init_states is None:
            h_t, c_t = (torch.zeros(bs, self.hidden_size).to(x.device),
                        torch.zeros(bs, self.hidden_size).to(x.device))
        else:
            h_t, c_t = init_states
        HS = self.hidden_size
        for t in range(seq_sz):
            x_t = x[:, t, :]
            # batch the computations into a single matrix multiplication
            gates = x_t @ self.W + h_t @ self.U + self.bias
            i_t, f_t, g_t, o_t = (
                torch.sigmoid(gates[:, :HS]), # input
                torch.sigmoid(gates[:, HS:HS*2]), # forget
                torch.tanh(gates[:, HS*2:HS*3]),
                torch.sigmoid(gates[:, HS*3:]), # output
            )
            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)
            hidden_seq.append(h_t.unsqueeze(0))
        hidden_seq = torch.cat(hidden_seq, dim=0)
        # reshape from shape (sequence, batch, feature) to (batch, sequence,
            feature)
        hidden_seq = hidden_seq.transpose(0, 1).contiguous()
        return hidden_seq, (h_t, c_t)
```

(二) 修改层数量，隐状态和通道数量

实验当中，我们还修改了层数量，隐状态和通道数量，得到了更多的结果。

增加 RNN 的层数可以使模型具有更高的复杂性，能够学习更复杂的模式。然而，这也可能导致过拟合，特别是当数据集较小时。此外，更深的 RNN 更难以训练，因为它们可能会遇到梯度消失或梯度爆炸的问题。本次实验当中，我们刚开始准确率有一定的增长，准确率开始不再有明显变化。

增加隐藏状态的数量：增加隐藏状态的数量可以增加模型的容量，使其能够存储和处理更多的信息。然而，这也可能导致过拟合，并增加训练和推理的计算成本。本次实验我们选取的模型容量小，使用隐状态优化效果明显。

改变通道数量：在处理图像或音频等多通道数据时，增加通道数量可以帮助模型捕捉到更多的特征。然而，这也会增加模型的复杂性和计算成本。本次实验当中测试了加倍和减半通道数量，结果并不是十分友好，说明初始模型就较为准确。

(三) 结果图形展示

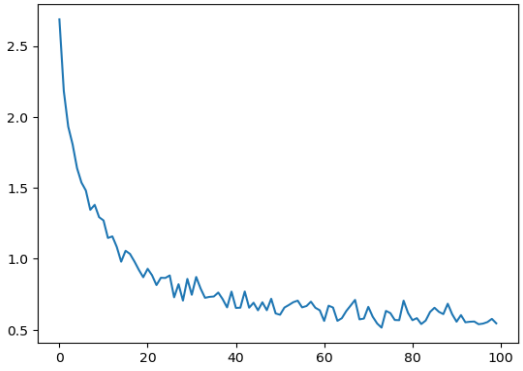

图 8: LSTM validation loss

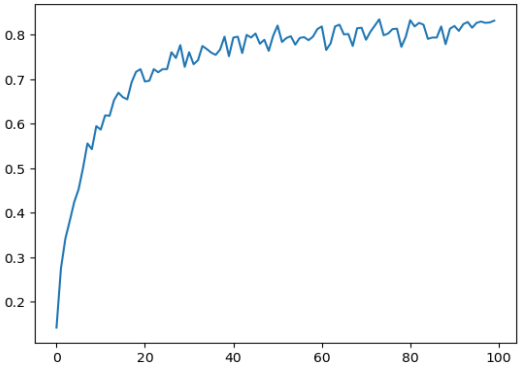

图 9: LSTM validation accuracy

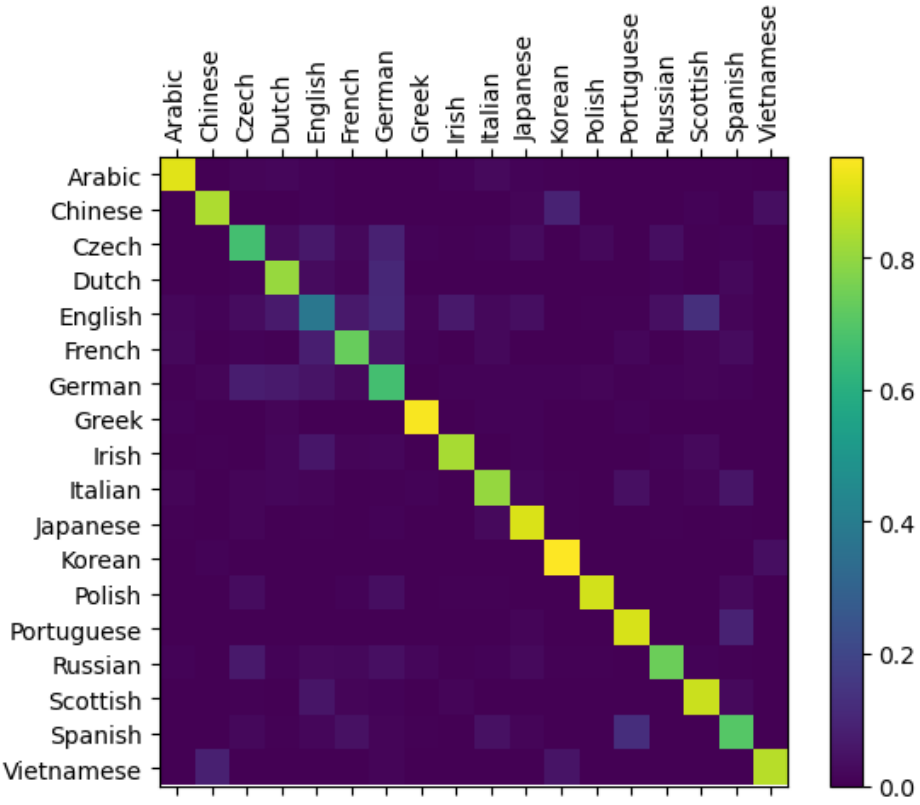

图 10: LSTM 预测矩阵

五、 总结

本次实验主要复现了两种循环神经网络，并进行了自主编写，循环神经网络是重要的深度学习分支，在编写的过程当中我感到受益匪浅。还了解了 wordvec 的相关知识，强化了对 NLP 的理解。同时实验过程当中我们还改变了不同的参数，给实验结果带来了一定的优化。希望未来学习的道路更加有信心，学习的更多有用的深度学习知识。