# 深度学习 Lab2 实验报告

PB19151769　　马宇骁

## 1　实验要求

编写 RNN 的语言模型，并基于训练好的词向量，编写 RNN 模型用于文本分类。

- 实现 SimpleRNN, LSTM, GRU(基于训练好的词向量 Glove)，并对结果进行对比;
- 对比单层 RNN 与两层 RNN;
- 对比隐藏神经元数量对结果的影响.

## 2　实验配置

由于实验本身并不十分复杂，但是环境配置调试过程过于艰辛，心力憔悴，故决定做环境配置过程几个关键点的回忆记录分析 (从决定使用 Old 版本的 ipynb 开始)。

### 2.1　本人电脑的“Lab2”的 conda 环境搭建

将助教的 requirement 在 Anaconda 创建好 conda 环境并将 utils.py 的 114 行改为 spacy = spacy.load(”en_core_web_sm”) 后，查看该环境的 CUDA 的支持版本，输出如下：

1.10.2+cpu

于是，决定将自己电脑的 CUDA 版本装为相应版本 10.2，并匹配对应的 CuDNN，将相关文件转移至 CUDA 安装文件夹下对应的位置。

在 cmd 中输入 nvcc -V 结果显示成功安装了 10.2 版本的 CUDA。测试是否能在 Lab2 中使用 CUDA，print(torch.cuda.is_available())，返回：

False

证明在该环境中暂时还是无法使用 GPU，考虑可能是 torch 的 cudatoolkit 可能一些细节不匹配，于是在 conda 环境中使用 conda install pytorch torchvision cudatoolkit=10.2（因为 pytorch 官网已经不支持 11 以下的安装，多次尝试在镜像中相似的各种装库方式（包含删库重装等）），最终，发现，不行。

最终在多次修改删除等操作之后，可能是不小心删除了不该删除的文件，在我的 VScode 的 jupyternotebook 中内核出现了问题。于是决定使用华为云的服务器进行实验。

### 2.2　华为弹性云服务器 1

根据文档提示，选择搭建 GPU 型的服务器默认第一个，内存 32GB，存储 40GB，Windows 系统。在经历了开机之后在 win+R 中 mstsc 远程连接相应的 IP 地址出现连接不上的问题后，

尝试网页版发现延迟太高，故选择解决远程连接问题，尝试各种办法之后，终于发现，如果电脑的系统是：

**Windows 10 家庭版**

那么，就不可能连接的上——它不支持远程连接。于是，升级了专业版。

解决连接问题之后进入服务器，安装 Anaconda 成功，在安装 Nvidia 驱动的时候，在安装快结束的时候，弹出了这么一个信息：

计算机丢失 wlanapi all,wlanapi.dll

经查询，wlanapi.dll 是电脑文件中的 dll 文件（动态链接库文件）。如果计算机中丢失了某个 dll 文件，可能会导致某些软件和游戏等程序无法正常启动运行，并且导致电脑系统弹窗报错。因此，考虑到是新电脑，决定在网上寻找该文件下载至 system32 文件夹中。再次安装，发现出现与链表有关的错误，于是在系统变量中添加相应路径再次尝试安装，依旧报错。

当我在点开系统属性之前我的解决方式均不生效，但当点开发现，这台服务器的系统是：

Windows Server 2019

前面的问题的解决方案立刻出现——装 Win Server2019 即选择 Tesla 驱动，而不是 win10 版的。于是，安装成功。

由于选择驱动的时候发现该版本只能选择 CUDA11 以上的版本，于是在安装 CUDA 的时候，选择 CUDA11.0。在下载好相应版本安装时，在安装的过程中，报出计算机空间不足后，尝试删除之前的一些安装包以及从云盘下载下来的 Lab2 文档中的数据集的压缩包等文件，再次安装。空间仍然不足，于是，在多次寻找删除无用文件但均不能将空间释放到安装 CUDA 需要的空间后，查询服务器信息发现只能扩展运存和 GPU 的数量，立刻，决定删除该服务器，选择重新搭建 100GB 的存储的服务器。

## 2.3 华为弹性云服务器 2

重复之前的正确的操作，由于助教提供的 CUDA 支持为 10.2，于是，考虑之前的尝试之后决定通过 pytorch 官网的加载方式，通过 pip 指令添加（但只有 11.3 的版本，不知道是否向下兼容，还是考虑可能出错），语句如下：

pip3 install torch torchvision torchaudio –extra-index-url https://download.pytorch.org/whl/cu113

等待，最终安装成功。测试是否 available，返回：

True

泪流满面。

关于将该环境添加到 jupyternotebook 中，由于之前犯过错误，于是决定在这里记录正确方式：

- conda install nb_conda
- conda install ipykernel
- conda install -n 环境名称 ipykernel
- python -m ipykernel install –user –name 环境名称

以此在环境中运行。

### 2.4 总结

1. 不要重复造轮子；
2. 网上的教程别全信，可能问题原因有区别；
3. 能相信官网就相信官网；
4. 长眼睛；
5. 千万记得别省不该省的。

# 3 模型

## 3.1 函数

### 3.1.1 torch.nn.RNN/LSTM/GRU()

参数：

- input_size 输入特征的维度，一般 rnn 中输入的是词向量，那么 input_size 就等于一个词向量的维度
- hidden_size 隐藏层神经元个数，或者也叫输出的维度（因为 rnn 输出为各个时间步上的隐藏状态）
- num_layers 网络的层数
- nonlinearity 激活函数
- bias 是否使用偏置
- batch_first 输入数据的形式，默认是 False，就是这样形式，(seq(num_step), batch, input_dim)，也就是将序列长度放在第一位，batch 放在第二位
- dropout 是否应用 dropout, 默认不使用，如若使用将其设置成一个 0-1 的数字即可
- birdirectional 是否使用双向的 rnn，默认是 False

在前向计算后会分别返回输出和隐藏状态 h，其中输出指的是隐藏层在各个时间步上计算并输出的隐藏状态，它们通常作为后续输出层的输 。需要强调的是，该“输出”本身并不涉及输出层计算，形状为 (时间步数, 批量大小, 隐藏单元个数)；隐藏状态指的是隐藏层在最后时间步的隐藏状态：当隐藏层有多层时，每 层的隐藏状态都会记录在该变量中；对于像短期记忆（LSTM），隐藏状态是 个元组 (h, c)，即 hidden state 和 cell state (此处普通 rnn 只有一个值) 隐藏状态 h 的形状为 (层数, 批量大小, 隐藏单元个数).

### 3.1.2 torch.nn.Embedding

参数：

- num_embeddings (python:int) –词典的大小尺寸，比如总共出现 5000 个词，那就输入 5000。此时 index 为（0-4999）

- embedding_dim (python:int) –嵌入向量的维度，即用多少维来表示一个符号。
- padding_idx (python:int, optional) –填充 id，比如，输入长度为 100，但是每次的句子长度并不一样，后面就需要用统一的数字填充，而这里就是指定这个数字，这样，网络在遇到填充 id 时，就不会计算其与其它符号的相关性。（初始化为 0）
- max_norm (python:float, optional) –最大范数，如果嵌入向量的范数超过了这个界限，就要进行再归一化。
- norm_type (python:float, optional) –指定利用什么范数计算，并用于对比 max_norm，默认为 2 范数。
- scale_grad_by_freq (boolean, optional) –根据单词在 mini-batch 中出现的频率，对梯度进行放缩。默认为 False.
- sparse (bool, optional) –若为 True, 则与权重矩阵相关的梯度转变为稀疏张量。

将文字转换为一串数字。因为数字是计算机更容易识别的一种表达形式。我们词嵌入的过程，就相当于是我们在给计算机制造出一本字典的过程。计算机可以通过这个字典来间接地识别文字。词嵌入向量的意思也可以理解成：词在神经网络中的向量表示。

模块的输入是一个索引列表，输出 shape =（*，H），其中 * 为输入的 shape，H = embedding_dim（若输入 shape 为 N*M，则输出 shape 为 N*M*H）；torch.nn.Embedding 的权重为 num_embeddings * embedding_dim 的矩阵，例如输入 10 个词，每个词用 3 为向量表示，则权重为 10* 3 的矩阵

## 3.2 类的构建

在此，由于实验要求，在此构建 3 个类：class RNN(nn.Module)，class LSTM(nn.Module)，class GRU(nn.Module)。以 RNN 类为例：

```
class RNN(nn.Module):
def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim, num_layers=1):
    super(RNN, self).__init__()
    self.embedding = nn.Embedding(input_dim, embedding_dim)
    self.rnn = nn.RNN(input_size = embedding_dim,
                      hidden_size = hidden_dim,
                      num_layers = num_layers)
    self.out = nn.Linear(hidden_dim, output_dim)
    self.sig = nn.Sigmoid()

def forward(self, text, text_length):
    text = self.embedding(text)
    text_length = text_length.cpu() # 拿回cpu
    text = pack_padded_sequence(text, text_length) #input: pad_sequence 的结果
                                                   length: mini-batch 中句子的实际
                                                   长度
    output, hidden = self.rnn(text)
    hidden = hidden[-1]
    output = self.out(hidden)
    output = self.sig(output)
    return output
```

思路即：先词嵌入，然后用 rnn 构建循环神经网络，选择最后 1 一个时刻的 RNN，由于是二分类，经过 Sigmoid 处理之后输出。（在 LSTM 的时候由于返回值的区别，将第 15 行的第二个传值改为一个 tuple 第一个仍为后续处理使用参数）

## 3.3 训练

training 过程形似现有网络上各种已有的思路，只需要将返回的参数修改添加以便后续更为直观展现不同网络的训练过程。

## 3.4 结果与分析

### 3.4.1 训练测试结果

对于 256 隐藏，单 RNN；256 隐藏，2RNN；512 隐藏，单 RNN；1024 隐藏，单 RNN；256 隐藏，LSTM；256 隐藏，GRU，总共 6 种网络进行训练，记录其训练过程每一次 epoch 的准确率和损失值。测试结果依次如下：

Test Loss: 0.584 | Test Acc: 69.76%

Test Loss: 0.572 | Test Acc: 71.35%

Test Loss: 0.592 | Test Acc: 69.03%

Test Loss: 0.622 | Test Acc: 65.74%

Test Loss: 0.389 | Test Acc: 83.08%

Test Loss: 0.395 | Test Acc: 83.26%

训练过程的准确率和损失用折线图绘制如下（由于服务器上在运行 matplotlib.pyplot 的时候总是内核重连，于是将所需数据导出为 data.csv 再在自己电脑上绘制）：

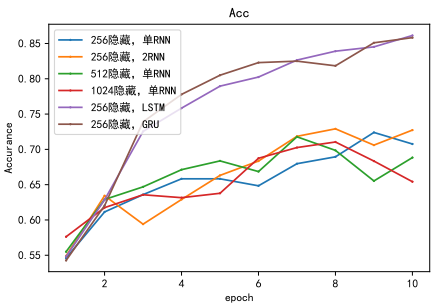


图 1: 准确率

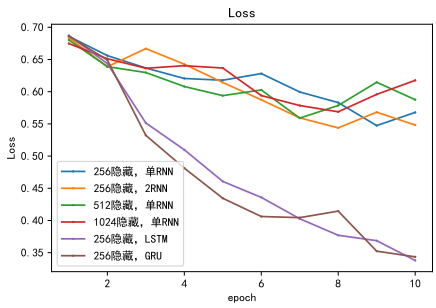


图 2: 损失率

### 3.4.2 分析讨论

- RNN 与 LSTM 与 GRU：

RNN(图3) 就是将网络的输出保存在一个记忆单元中，这个记忆单元和下一次的输入一起进入神经网络中。RNN 对所有输入具有同等的特征提取的能力，对于一句话中的所有单词，都会提取他们的特征，并作为下一特征的输入。也就是说 RRN 网络，记住了所有的信息，并没有区分哪些是有用信息，有哪些是无用信息，哪些是辅助信息。

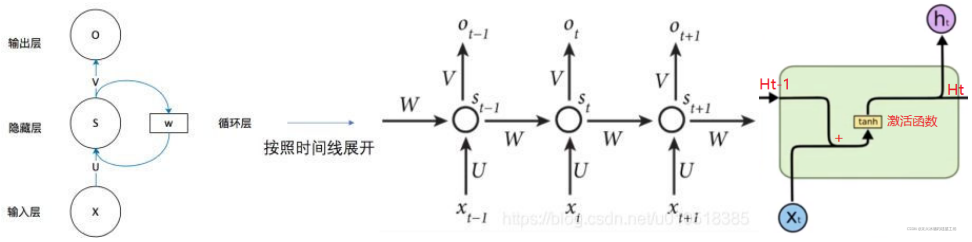


图 3: RNN

然而，实际语言文字中，不同单词虽然有特征，但对目标的作用是不同的，有些词，对于最终的目标，其实没有任何意义，就是修饰词而已。而有些词的特征，起着决定性对的作用，这些词，就需要通篇文章范围内进行记忆，也就是说，不同词的重要性，不仅仅取决于词之间的物理距离，而且取决于词本身的意义。

在该结构中，只有一个记忆状态 Ht-1，该记忆信息与当前的输入信息一起，产生一个新的记忆状态 Ht。所谓记忆，这里就是 Ht-1、Ht 的值。而 H 对应的权重矩阵，就是对记忆的提取。所谓当前的输入，就是 Xt。而与 Xt 对应的权重矩阵，就是瞬时记忆。

在这里，Ht-1 是所有历史输入的记忆汇总，即无丢弃，无选择性记忆。历史输入离当前的时间越远，对当前输出 Ht 的影响越小，即长期记忆被遗忘。而 Ht = Tanh(Xt*Wx + Ht-1 * Wh)，是历史记忆 + 瞬时记忆产生新的记忆。

LSTM（long short-term memory）长短期记忆网络是 RNN 的一种变体，LSTM 网络通过精妙的门控制将短期记忆与长期记忆结合起来，并且一定程度上解决了梯度消失的问题。LSTM 的 Cell 是一个具备选择性记忆功能的记忆单元。

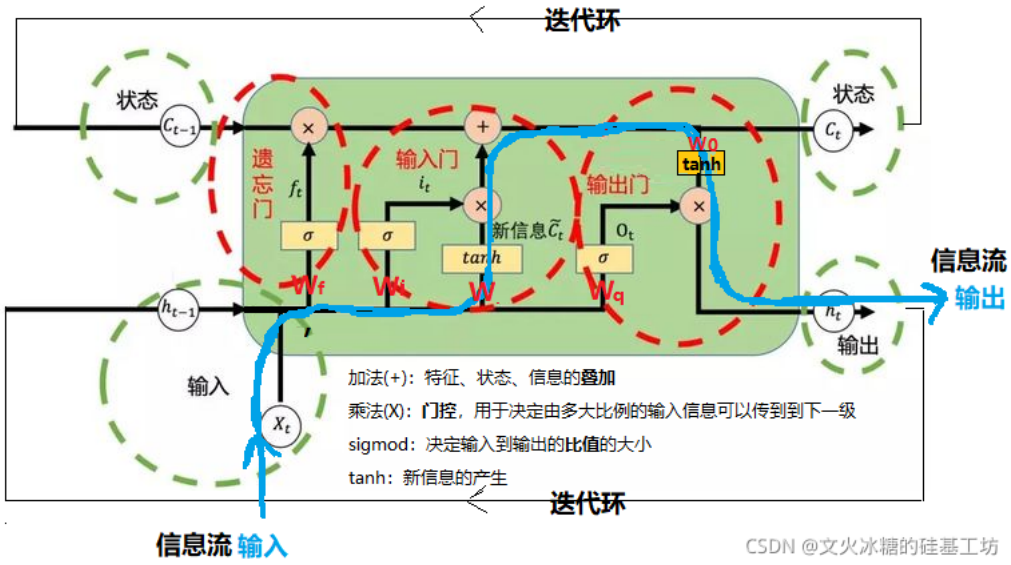


图 4: LSTM

GRU 和 LSTM 最大的不同在于 GRU 将遗忘门和输入门合成了一个” 更新门”，同时网络不再额外给出记忆状态，而是将输出结果作为记忆状态不断向后循环传递，网络的输人和输出都变得特别简单。

当时间步较大或者较小时，RNN 容易出现梯度衰减或者梯度爆炸。虽然梯度裁剪可以应对梯度爆炸，但是无法解决梯度衰减的问题。通常由于这个原因，RNN 在实际中较难捕捉时

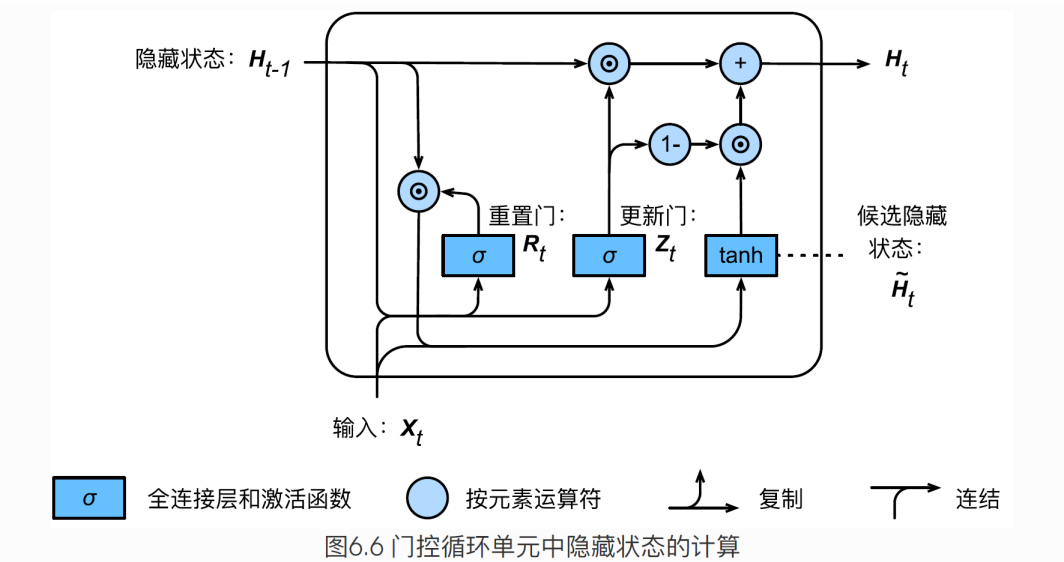


图 5: GRU

间序列中时间步距离较大的依赖关系。门控循环神经网络 (gated recurrent neural network) 的提出，正是为了更好地捕捉时间序列中的时间步较大的依赖关系。它可以通过学习的门来控制信息的流动。其中，门控循环单元 (gated recurrent unit, GRU) 是一种常用的门控循环神经网路。

由此，对比曲线（图1,2）和 1、5、6 测试准确率，能够证明 LSTM 和 GRU 对于 RNN 的改进。

- RNN 单层与双层

双层 RNN 顾名思义，即 RNN 之间有一次嵌套关系。输入数据整体上是一个时间序列，而对于每一个内层特征数据而言，也是一个时间序列。即二维数组，或者数组的数组这个概念。而双层 RNN 是可以处理这种输入数据的网络结构。

但该数据的结果显示，RNN 单双层数对于准确率的影响并不大。

- 隐藏层数量

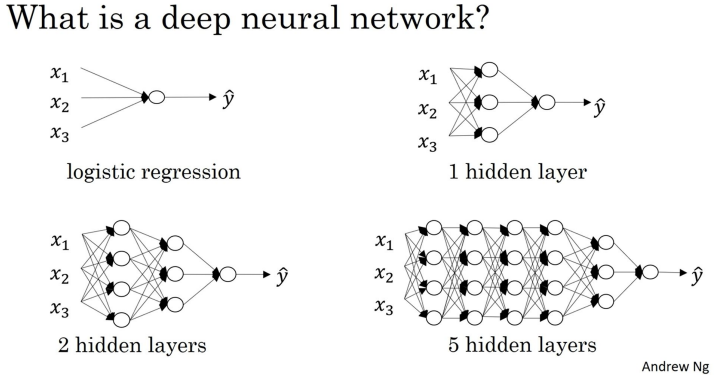


图 6: 隐藏层

层数越深，理论上拟合函数的能力增强，效果按理说会更好，但是实际上更深的层数可能会带来过拟合的问题，同时也会增加训练难度，使模型难以收敛。有结果，对于该数据集，256 层以上的隐藏层的效果随着层数增加而倒退。