实验四：循环神经网络

李旻翀

1190200208

日期：June 12, 2022

摘　　要

本实验为模式识别与深度学习课程的实验四：循环神经网络。在本次实验中，我利用 Pytorch 自己实现了 RNN、GRU、LSTM 和 Bi-LSTM 四种网络的结构，并利用自己实现的上述网络结构，完成了文本多分类和温度预测的任务。最终，我在文本多分类任务上可以达到 0.88 的准确率，而采用 GRU 结构可以在温度预测任务上也取得了较好的效果。

关键词：模式识别，循环神经网络，RNN，GRU，LSTM，Bi-LSTM，文本多分类，温度预测

1　网络结构实现

在本次实验中，我利用 Pytorch 自己实现了 RNN、GRU、LSTM 和 Bi-LSTM 四种网络的结构。具体而言，网络结构实现分为两部分，一部分为实现循环神经网络单个 cell 的结构（代码保存在 RNNcell_series.py 中），另一部分为将 cell 组合为完整的网络（代码保存在 RNN_series.py 中）。

下面以 RNNcell 和 RNN 结构为例，说明我所实现的网络。

```python
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers=1, bias=True, act_fn='tanh'):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.bias = bias

        self.rnn_cell_list = nn.ModuleList()  # 与nn.Sequential类似，区别是nn.Sequential无法自定义forward函数（它会自动构建）

        self.rnn_cell_list.append(RNNcell(self.input_size, self.hidden_size, self.bias, act_fn))

        for i in range(1, self.num_layers):
            self.rnn_cell_list.append(RNNcell(self.hidden_size, self.hidden_size, self.bias, act_fn))
```

图 1: RNNcell 结构

在我所实现的 RNNcell 中，可以选择 tanh 或者 Relu 作为激活函数（默认为 tanh），并按照 RNN 的结构重写了 forward 方法。而整体的 RNN 网络被设定为多个 RNNcell 的结合，可以根据指定的模型层数搭建多层 RNN。在 RNN 网络的 forward 函数中，可以读入前一个 RNNcell 的输出，从而实现“记忆”功能。

我实现的四种网络接口与 torch.nn 中对应的网络模型接口完全相同，在实际搭建的网络中可以直接替换为 torch.nn.RNN 等模型。

2 文本多分类任务

2.1 数据预处理

相较于计算机视觉方面的任务，文本任务需要进行复杂的预处理。预处理相关的代码保存在./utils 下，以下按文件运行顺序简述预处理过程：

2.1.1 build_words.py

这一部分完成的任务有：

1. 读入 online shopping 数据集
2. 将 label 映射为 number
3. 调用 jieba 库对文本进行分词
4. 完成训练集、测试集、验证集的划分
5. 将分好词的训练集、测试集、验证集保存到本地（按照“标签, 文本”的格式）

运行 buildwords.py 文件后，可以在./data/online_shopping 下得到 (train/test/val).words.txt 文件，其样式如下：

0, 做 父母 一定 要 有 刘墉 这样 的 心态 ， 不断 地 学习 ， 不断 地 进步 ， 不断 地 给 自己 补充 新鲜血液 ， 让 自己 保持 一颗 年轻 的 心 。 我 想 ， 这 是 他 能 很 好 的 和 孩子 沟通 的 一个 重要 因素 。 读 刘墉 的 文章 ， 总能 让 我 看到 一个 快乐 的 平易近人 的 父亲 ， 他 始终 站 在 和 孩子 同样 的 高度 ， 给 孩子 创造 着 一个 充满 爱 和 自由 的 生活 环境 。 很 喜欢 刘墉 在 字里行间 流露出 的 做 父母 的 那种 小 狡黠 ， 让 人 总是 忍俊不禁 ， 父母 和 子女 之间 有时候 也 是 一种 战斗 ， 武力 争斗 过于 低级 了 ， 智力 较量 才 更 有 趣味 。 所以 ， 做 父母 的 得 加把劲 了 ， 老 思想 老 观念 注定 会 一败涂地 ， 生命不息 ， 学习 不止 。 家庭教育 ， 真的 是 乐在其中 。
0,作者 真有 英国人 严谨 的 风格 ， 提出 观点 、 进行 论述 论证 ， 尽管 本人 对 物理学 了解 不深 ， 但是 仍然 能 感受 到 真理 的 火花 。 整本书 的 结构 颇 有 特点 ， 从 当时 （ 本 书写 于 八十年代 ） 流行 的 计算机 话题 引入 ， 再用 数学 、 物理学 、 宇宙学 做 必要 的 铺垫 — — 这些 内容 占据 了 大部分 篇幅 ， 最后 回到 关键问题 ： 电脑 能 不能 代替 人脑 。 和 现在 流行 的 观点 相反 ， 作者 认为 人 的 某种 “ 洞察 ” 是 不能 被 算法 模拟 的 。 也许 作者 想 说 ， 人 的 灵魂 是 无可取代 的 。

图 2: build_words.py 处理结果

2.1.2 build_vocab.py

这一部分完成的任务是按照词的顺序构建词典,运行 build_vocab.py 后,在./data/online_shopping 下得到 vocab.words.txt 文件，其样式如下：

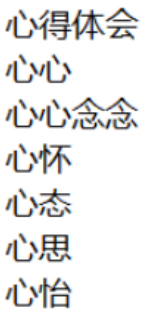

图 3: build_vocab.py 处理结果

需要注意的是，**这一部分只能根据训练集来构建词表**。如果使用所有数据构建词表，则会出现“标签泄露”的情况（即：在测试时所有词在 embedding 矩阵中都有对应的向量，不会被归为 <UNK>）。

2.1.3 build_embeddings.py

这一部分完成的任务是初始化 embedding 矩阵，为 RNN 模型的 embedding 层提供初始化权重。在这一步中，我使用了开源词向量Chinese-Word-Vectors所提供的 sgns.merge.bigram。

在这一步中，embedding 矩阵的大小为 (vocab size +1, 300)，对其的解释如下：

1. vocab size +1：代表 vocab 表中出现的所有词都有自己对应的词向量，多余的 1 表示 <UNK> 和 <PAD>（为了简便，我把这两类标签归为一条向量，后续来看对结果影响不大）。
2. 300：代表词向量的维度，即一个词被表示为 300x1 的向量。设为 300 是因为开源词向量Chinese-Word-Vectors的维度是 300。

对于词表中的词，将其词向量置为在开源词向量Chinese-Word-Vectors中对应的 300 维向量，若其未出现在开源词向量Chinese-Word-Vectors中，则将其对应的 300 维词向量全置为 0。标签 <UNK> 和 <PAD> 对应着 embedding 矩阵的第一行，也全置为 0。

运行 build_embeddings.py 后，初始化的词向量权重保存在./data/online_shopping/w2v.npz 中。

2.1.4 load_online_shopping.py

这一部分完成的任务是按照 max len 切分每一段文本（只保留从头开始的最多 max len 长度的文本），并将文本转化为文本在 embedding 矩阵中对应的行号（也即是在 vocab 中对应的行号）。完成后，将数据装入 dataloader。

2.2 网络结构

用于文本多分类的网络结构如下：

```python
class OnlineShoppingNet(nn.Module):
    def __init__(self, embeddings, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding.from_pretrained(embeddings=embeddings, freeze=False, padding_idx=0)
        self.rnn = RNN_series.RNN(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)
```

图 4: 用于文本多分类的 RNN 结构

其包含三层：embedding 层，RNN 层，以及最后的全连接层。embedding 层使用了在上一步中导入的预训练权重，RNN 层是我自行实现的循环神经网络结构。全连接层用于归一化结果。

最终，模型的输入为 [batch size(64), max len(50)]。经过 embedding 层后，实际输入 RNN 层的向量为 [batch size(64), max len(50), embedding dim(300)]。

2.3 训练过程

文本多分类任务的训练过程保存在 train.py 中，主要模块（如模型训练，测试，反向传播）来自于前几次实验。本实验中，我使用了 sklearn 库中的 classification_report 方法，可以直接计算模型在测试集上的准确率，召回率与 F1 值。

对于每个模型，进行 60 个 epoch 的训练，学习率设置为 0.0001，优化器采用 Adam，损失函数为交叉熵函数。

2.4 实验结果

以 LSTM 模型为例，其训练与评测结果如图所示：

```
Epoch:55/60   Train Loss: 0.0391   Val Loss: 0.6540   Acc:0.8727
Epoch:56/60   Train Loss: 0.0409   Val Loss: 0.6204   Acc:0.8732
Epoch:57/60   Train Loss: 0.0394   Val Loss: 0.6825   Acc:0.8719
Epoch:58/60   Train Loss: 0.0342   Val Loss: 0.6897   Acc:0.8673
Epoch:59/60   Train Loss: 0.0320   Val Loss: 0.6989   Acc:0.8725
Epoch:60/60   Train Loss: 0.0347   Val Loss: 0.6480   Acc:0.8728
Finished 60 epoch
Test Acc:0.8803   Test Loss:0.6199
```

图 5: LSTM 训练结果

图 6: LSTM 评估结果

四个模型的实验结果如下图（相关图片文件保存在根目录的./results 下）:

表 1: 实验结果

	precision	**recall**	**f1-score**	**accuracy**
RNN	0.73	0.71	0.72	0.80
LSTM	0.87	0.86	0.86	0.88
GRU	0.87	0.83	0.85	0.87
Bi-LSTM	0.86	0.86	0.86	0.88

从实验结果可以看出，LSTM，GRU，Bi-LSTM 的结果差距不大，而 RNN 的训练效果差于前三者。此外，GRU 由于其结构较 LSTM 更为简单，简化了计算量，因此在运行时速度最快。

3 温度预测任务

3.1 数据预处理

温度预测任务的数据预处理程序保存在 load_jena.py 中，主要处理步骤如下:

1. 处理日期数据。通过 datetime.strptime() 方法将原本的字符串转化成 datetime 类型的数据，便于后续处理。
2. 将日期数据转化为正余弦函数的数值形式，使得模型在训练时能够感知到时间周期。
3. 对数据集进行归一化，剔除无用数据（只保留 month, day 对应的 sin, cos 值，以及温度，共 5 个特征）。
4. 按前 5 年为一段，后 2 年为一段划分数据集。
5. 根据原始数据数据集得到模型的输入与输出。并将数据装入 dataloader。按照实验指导书的要求，需要以 5 天的温度值预测 2 天的温度值。因此，模型的输入为 5 天的温度值及时间数据（720x5 的 tensor），输出为 2 天的温度值（288x1 的 tensor）。

3.2 网络结构

用于温度预测的网络结构如下：

```python
class JenaNet(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.rnn = RNN_series.GRU(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, data):
        output, hidden = self.rnn(data)

        a = output[:, -1, :]
        b = hidden.squeeze(0)

        assert torch.equal(output[:, -1, :], hidden.squeeze(0))

        return self.fc(hidden.squeeze(0))
```

图 7: 用于温度预测的 GRU 结构

由于不涉及词与词向量的映射，因此去掉了 embedding 层，直接输入 [batch size(16), data of 5 days(720), feature num(5)] 的 tensor。

3.3 训练过程

温度预测任务的训练过程保存在 train_jena.py 中。进行 100 个 epoch 的训练，学习率设置为 0.0001，优化器采用 Adam，损失函数为 MSE。

由于温度预测是预测任务，而不是前几个实验所做的的分类任务，因此，需要对损失函数，test() 函数进行一系列修改，此处不再赘述。

3.4 实验结果

开始训练后，train loss 和 test loss 迅速下降，训练速度也非常快，最终得到的结果如下：

随机选取某一段的预测值与真实值作图，结果如下：

```
Epoch:90/100   Train Loss: 0.1293   Val Loss: 0.1258
Epoch:91/100   Train Loss: 0.1291   Val Loss: 0.1346
Epoch:92/100   Train Loss: 0.1271   Val Loss: 0.1327
Epoch:93/100   Train Loss: 0.1296   Val Loss: 0.1248
Epoch:94/100   Train Loss: 0.1254   Val Loss: 0.1350
Epoch:95/100   Train Loss: 0.1269   Val Loss: 0.1315
Epoch:96/100   Train Loss: 0.1287   Val Loss: 0.1186
Epoch:97/100   Train Loss: 0.1268   Val Loss: 0.1333
Epoch:98/100   Train Loss: 0.1306   Val Loss: 0.1284
Epoch:99/100   Train Loss: 0.1270   Val Loss: 0.1325
Epoch:100/100   Train Loss: 0.1269   Val Loss: 0.1311
Finished 100 epoch
Test Loss:0.1184
Avg error: -0.1804
Median error: -0.1602
Figure shows the prediction on certain period of test set.
```

图 8: 温度预测训练结果

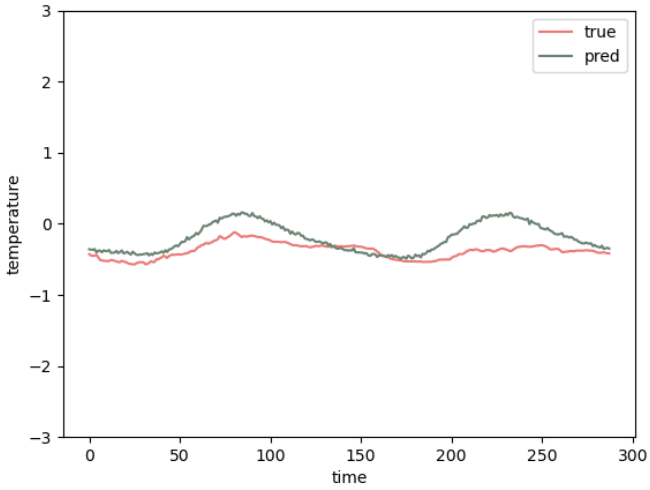

图 9: 温度预测结果

4 说明文档

4.1 实验文件夹结构

- custom_dataset.py：自定义的 online shopping 和 jena climate 数据集
- JenaNet.py：用于温度预测的网络
- OnlineShoppingNet.py：用于文本多分类的网络
- RNN_series.py：自己实现的循环神经网络结构
- RNNcell_series.py：自己实现的循环神经网络单个单元的结构
- train.py：文本多分类的训练程序文件
- train_jena.py：温度预测的训练程序文件
- ./data：保存 online shopping 和 jena climate 数据集
- ./model：保存训练好的模型（本实验中仅保留最优模型）

- ./utils：保存训练程序中用到的功能函数
- ./runs：保存文本多分类的 tensorboard 日志文件
- ./runs_jena：保存温度预测的 tensorboard 日志文件