实验2-循环神经网络实现

符兴 7203610316

**1 实验环境**

Ubuntu 20.04，Python 3.9， PyTorch1.13，Cuda11.7，TensorBoard；

|  |
| --- |
| 1 |
| 图1 Python环境 |

**2 实验过程**

**2.1 模型构建**

在本次实验中，需要分别构建RNN、GRU、LSTM和BiLSTM网络的模型。

2.1.1 RNN模型

在本次实验中，基于nn.Linear()搭建了RNN的网络结构；同时，基于PyTorch中自带的SequencePacked结构，实现了多个语句并行计算；具体的计算步骤为:

(1)使用nn.utils.rnn中pack\_padded\_sequence()工具，将多个语句按列进行合并，该函数提供按列合并后每个batch\_size的长度以及内部语句的顺序；

(2)模型根据batch\_size的长度从输入中读取数据，然后根据sorted\_indices()提供的排列顺序迭代更新每一个时间步的hidden和output；

(3)在得到结果后，经过classfier层得到最终的输出；

|  |
| --- |
|  |
| 图2 RNN模型 |

2.1.2 LSTM模型

RNN中会出现梯度消失的问题，即近处梯度作为主导而远处的梯度逐渐消失，因此简单的RNN很难建模长距离的依赖关系。在此基础上，人们通过引入门机制来控制信息的传播。

在LSTM中，第一步由遗忘门通过读取当前输入x和前神经元信息h，由来决定从Cell State中丢弃什么信息，输出结果为0~1的值，表示保留该信息的程度；

第二步是确定Cell State所存放的新信息，sigmoid层决定将要更新的值；tanh层创建一个新的候选值向量加入到状态中。

第三步就是更新Cell State，将更新为 。把旧状态与相乘，接着加上。

最后一步先通过sigmoid来确定Cell State的哪个部分将输出出去。然后将Cell State进行tanh处理，得到一个在-1到1之间的值，并将它和sigmoid的输出相乘，得到最终的输出。

除此之外，LSTM还有一个双向的版本BiLSTM，BiLSTM由两个LSTM组合而成，一个是正向去处理输入序列，另一个反向处理序列，处理完成后将两个LSTM的输出拼接起来。

|  |
| --- |
|  |
| 图3 LSTM 模型 |

2.1.3 GRU模型

GRU作为LSTM的一种变体，将遗忘门和输入门合成为一个更新门，同时还混合了细胞状态和隐藏状态，最终的模型比标准的 LSTM 模型要简单，参数量更小，收敛速度更快。

GRU首先通过更新门控制当前状态需要从上一时刻状态中保留多少信息，以及从候选状态中接受多少信息；GRU通过重置门控制候选状态的计算。

|  |
| --- |
|  |
| 图4 GRU模型 |

**2.2 模型结构**

|  |
| --- |
|  |
| 图4 模型结构 |

本次实验是仿照AlexNet网络进行设计，大致分为两个部分，一个是基于卷积神经网络进行特征提取，另一个是根据特征进行分类的网络；

在特征提取层中，大致可以分为五个部分：

(1) 卷积层+池化层：卷积层的输出维度96，kernal\_size：11，步长为4，padding为0；池化层的kernal\_size为3，步长为2，padding为0；最后的输出为96\*26\*26；

(2) 卷积层+池化层：卷积层的输出维度256，kernal\_size：5，步长为1，padding为2；池化层的kernal\_size为3，步长为2，padding为0；最后的输出为256\*12\*12；

(3) 卷积层：卷积层的输出维度384，kernal\_size：3，步长为1，padding为1；最后的输出为384\*12\*12；

(4) 卷积层：卷积层的输出维度384，kernal\_size：3，步长为1，padding为1；最后的输出为384\*12\*12；

(5) 卷积层+池化层：卷积层的输出维度256，kernal\_size：3，步长为1，padding为1；池化层的kernal\_size为3，步长为2，padding为0；最后的输出为256\*5\*5；

在分类层中，大致可以分为几个部分：

(1) Dropout+Linear+ReLU：输出为4096\*1

(2) Dropout+Linear+ReLU：输出为2048\*1

(3) Dropout+Linear：输出为101\*1

同时，在本次实验中使用交叉熵作为损失函数，由于PyTorch所提供的计算交叉熵的CrossEntropyLoss()函数中已经集成了softmax，则在forward函数中不需要显式对结果做归一化处理；优化器选择Adam优化器，并且使用MultiStepLR对学习率根据epoch进行动态调整；

**3 实验结果**

(1) 没有数据增强的训练，在测试集上的测试结果为0.65

|  |
| --- |
| non |
| 图5 没有数据增强的训练结果 |

(2) 使用仿射变换数据增强的训练，在测试集上的测试结果为0.702

|  |
| --- |
| sher |
| 图6 在线数据增强的训练结果 |

(3) 使用多种数据增强+权重衰减的训练，在测试集上的测试结果为0.728

|  |
| --- |
| multi |
| 图7 多种数据增强+权重衰减的训练结果 |

(4) 只有权重衰减的训练，在测试集上的测试结果为0.71

|  |
| --- |
| none_weight |
| 图8 只有权重衰减的训练结果 |

通过上述四组实验数据的对比，可以发现，纯粹使用原有数据集进行训练的模型，在训练的后期出现了过拟合的情况，即训练集正确率仍在上升，但是验证集的正确率已趋于不变；当使用一定程度的数据增强时，模型的泛化能力有所增强，如图6所示，验证集的最高正确可以超过0.7，而在图5所示的数据中，验证集的最高正确率不超过0.7；

除此之外，本次实验还设置了权重衰减的实验组，如图7所示，在多种数据增强和权重衰减的训练中，验证集的最高正确率可以超过0.75，在测试集上的正确率也能达到0.728，相比图6的数据而言，模型的精度提高了7%左右；同时，单独设置了一个只有权重衰减的训练过程，其验证集最高精度不超过0.75，在测试集上的测试结果为0.71，低于图6的模型精度。

**4 心得与体会**

在本次实验中，发现在训练的过程中可以使用多种手段去提高模型的精度，如数据增强、权重衰减等；前者是通过增加数据的样本量达到提高模型泛化能力的目的；而后者是通过在反向传播的过程中，依据模型权重去限制损失函数的计算，进而避免模型过拟合；通过多次实验发现，数据增强的手段非常有效，通过对图像的翻转、仿射变换、截取以及增加噪声等等方式，模型可以学习到更多物体的特征，进而有更高的精度；