大作业实验报告

许明浒 2010984 信息安全法学双学位

一、实验任务

给定一个信息的标题、出处、相关链接以及相关评论，尝试判别信息真伪。

二、实验思路

采用深度学习方法，利用 LSTM 框架进行操作，分 4 个步骤

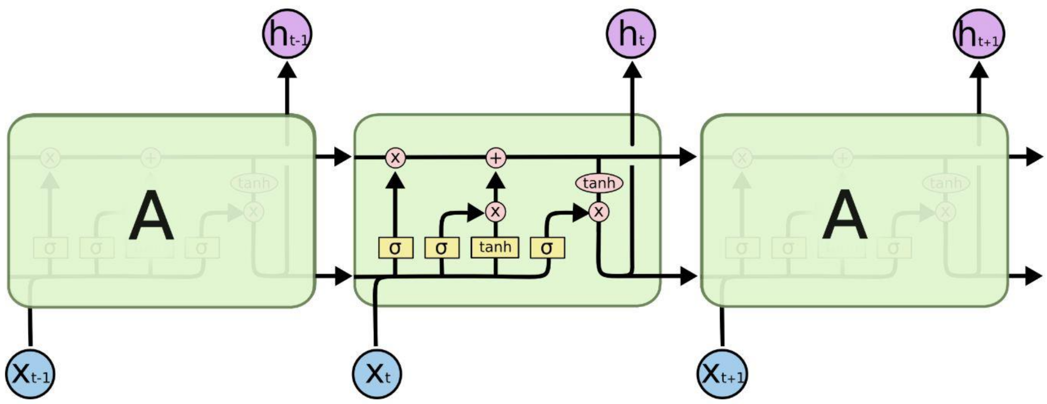
1.对数据进行预处理

2.准备深度学习所需要的数据

3.建立 LSTM 网络

4.训练与测试网络

三、LSTM 初步认识



The repeating module in an LSTM contains four interacting layers.

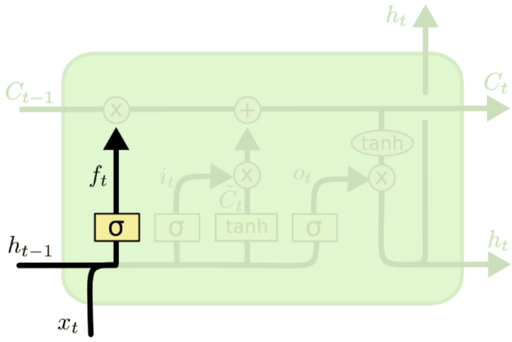
LSTM 是一种特殊的 RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的 RNN，LSTM 能够在更长的序列中有更好的表现。

两个传输状态，一个 h^t 和一个 c^t 。三个门：输入门、遗忘门、输出门。细胞状态 c^t 类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息

在上面流传保持不变会很容易。

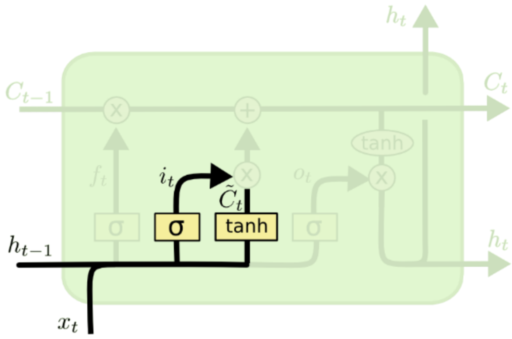
具体公式：

遗忘门丢弃信息



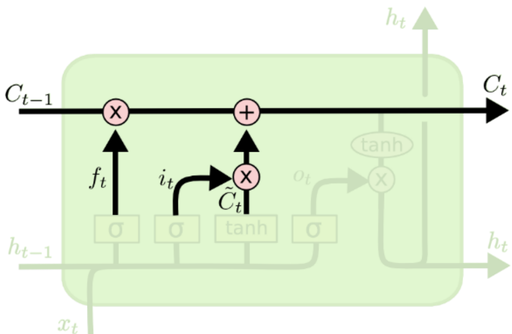
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

输入门确定更新的信息



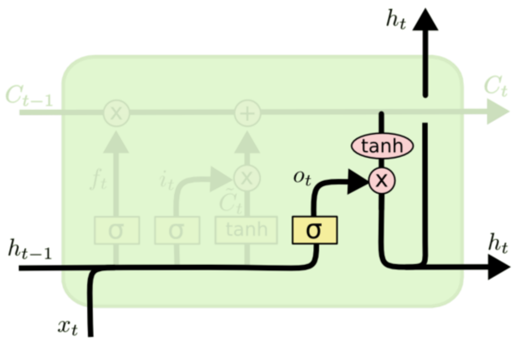
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

更新细胞状态



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

输出门输出信息



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

四、主要过程

1.文本预处理：

进行分词，去停用词等操作

```
with open(r"C:\Users\xumin\大作业/stop_words.utf8", encoding="utf8") as f:
    stopwords = f.readlines()
def Chinese_pre(text_data):
    #字母小写
    text_data = text_data.lower()
    #分词
    text_data = list(jieba.cut(text_data ,cut_all=False))
    #去停用词和多余空格
    text_data = [word.strip() for word in text_data if word not in stopwords]
    #处理后的词语使用空格连接为字符串
    text_data = " ".join(text_data)
    return text_data
```

打乱数据集并生成新的训练集和验证集

```
train_dataset["title"] = train_dataset["Title"].apply(Chinese_pre)
test_dataset["title"] = test_dataset["Title"].apply(Chinese_pre)
from sklearn.utils import shuffle
newtrain_dataset=shuffle(train_dataset)
valid_dataset=newtrain_dataset[9001:10586][["label","title"]]
train1_dataset=newtrain_dataset[0:9000][["label","title"]]
```

2.深度学习有关数据处理

主要利用 torchtext 库进行，构建新数据集，迭代器，词表

```
#构建数据集
mytokenize = lambda x: x.split()
TEXT = Field(sequential=True,tokenize=mytokenize,include_lengths=True,use_vocab=True,batch_first=True)
LABEL=Field(sequential=False,use_vocab=False,pad_token=None,unk_token=None)
text_data_fields = [("label",LABEL),("title",TEXT)]
traindata,validdata,testdata=TabularDataset.splits(path=r"C:\Users\xumin\大作业",format="csv",train="train2.csv",fields=text_data_fields,
                                                   validation="val2.csv",test="test2.csv",skip_header=True)
```

```
#构建迭代器
BATCH_SIZE = 64
train_iter = BucketIterator(traindata,batch_size = BATCH_SIZE)
val_iter = BucketIterator(validdata,batch_size = BATCH_SIZE)
test_iter = BucketIterator(testdata,batch_size = BATCH_SIZE)

#构建词表
TEXT.build_vocab(traindata,max_size=20000,vectors = None)
LABEL.build_vocab(traindata)
```

3. LSTM 网络

```python
class LSTMNet(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMNet, self).__init__()
        self.hidden_dim = hidden_dim #LSTM神经元个数
        self.layer_dim = layer_dim  #LSTM层数
        #对文本进行词向量处理
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        #LSTM+线性层
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, layer_dim, batch_first=True)
        self.fc1 = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        embeds = self.embedding(x)
        r_out, (h_n, h_c) = self.lstm(embeds, None)
        out = self.fc1(r_out[:, -1, :])
        return out
```

主要是三个层，embedding 层、LSTM 层、线性层。输入文本经过 embedding 层转化为词向量，进入 LSTM 层，进行数据计算，最后通过线性层，转化为要求的结果，即 0 或 1

超参的设置：

```python
TEXT.build_vocab(traindata, max_size=20000, vectors=None)
LABEL.build_vocab(traindata)
vocab_size = len(TEXT.vocab)
embedding_dim = 100
hidden_dim = 100
layer_dim = 1
output_dim = 2
lstmmodel = LSTMNet(vocab_size, embedding_dim, hidden_dim, layer_dim, output_dim)
lstmmodel
```

```python
optimizer = torch.optim.Adam(lstmmodel.parameters(), lr=0.0003)
loss_func = nn.CrossEntropyLoss()
lstmmodel, train_process = train_model2(lstmmodel, train_iter, val_iter, loss_func, optimizer, num_epochs=20)
```

4.训练和测试

训练：

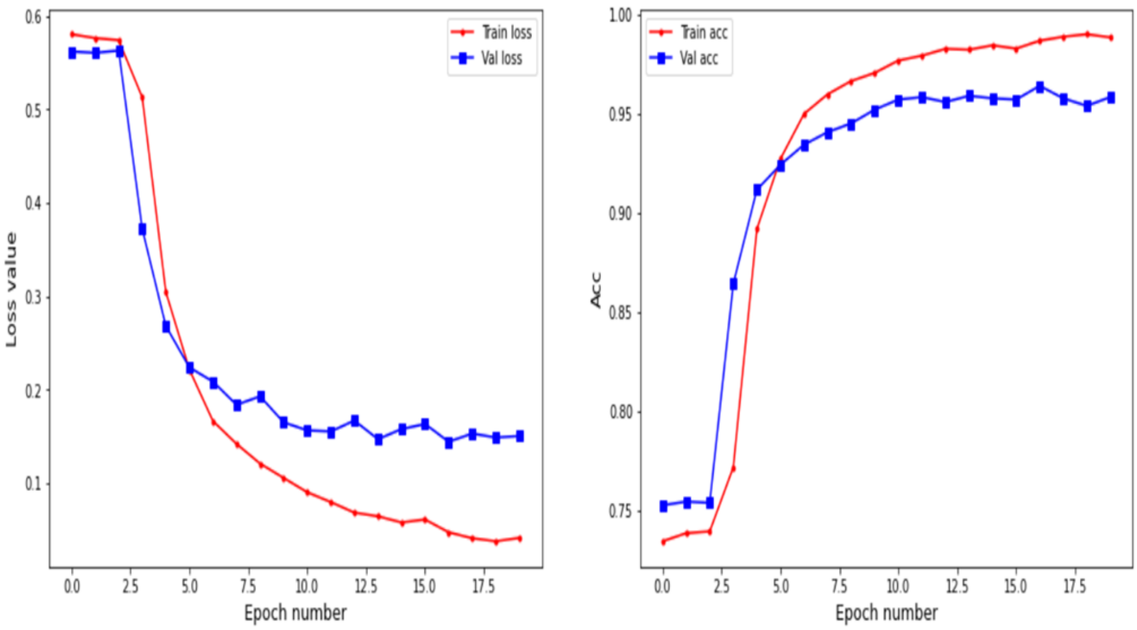
每次 epoch，模型先设置为训练阶段，计算 loss 和 acc，再设置为验证阶段，在验证集上验证模型的效果

```python
def train_model2(model,traindataloader,valdataloader,criterion,optimizer,num_epochs ):
    train_loss_all = []
    train_acc_all = []
    val_loss_all = []
    val_acc_all = []
    since = time.time ()
    for epoch in range(num_epochs):
        print('-'*10)
        print('Epoch{}/{}'.format(epoch,num_epochs-1))
        ##每个epoch有两个阶段
        train_loss =0.0
        train_corrects = 0
        train_num = 0
        val_loss =0.0
        val_corrects = 0
        val_num = 0
        model.train()
        for step,batch in enumerate(traindataloader):
            textdata,target = batch.title[0],batch.label.view(-1)
            out = model(textdata)
            pre_lab = torch.argmax(out,1)
            loss = criterion(out,target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```python
            loss = criterion(out,target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss+=loss.item()*len(target)
            train_corrects += torch.sum(pre_lab ==target.data)
            train_num += len(target)
        ##计算一个epoch在训练集上的损失和精度
        train_loss_all.append(train_loss / train_num)
        train_acc_all.append(train_corrects.double().item()/train_num)
        print('{}Train Loss:{:.4f}  Train Acc:{:.4f}'.format(
            epoch, train_loss_all[-1],train_acc_all[-1] ))
        #评估阶段
        model.eval()
        for step,batch in enumerate(valdataloader):
            textdata,target = batch.title[0],batch.label.view(-1)
            out = model(textdata)
            pre_lab = torch.argmax(out,1)
            loss = criterion(out,target)
            val_loss += loss.item()*len(target)
            val_corrects += torch.sum(pre_lab ==target.data)
            val_num += len(target)

        ##计算一个epoch在验证集上的损失和精度
        val_loss_all.append(val_loss / val_num)
        val_acc_all.append(val_corrects.double().item()/val_num)
        print('{}Val Loss:{:.4f}  Val Acc:{:.4f}'.format(
            epoch, val_loss_all[-1],val_acc_all[-1]))
    train_process = pd.DataFrame(
            data={"epoch":range(num_epochs),
                  "train_loss_all":train_loss_all,
                  "train_acc_all":train_acc_all,
                  "val_loss_all":val_loss_all,
                  "val_acc_all":val_acc_all   })
    return model,train_process
```

```
Epoch15/19
15Train Loss:0.0417  Train Acc:0.9880
15Val Loss:0.1743  Val Acc:0.9552
----------
Epoch16/19
16Train Loss:0.0359  Train Acc:0.9897
16Val Loss:0.1726  Val Acc:0.9457
----------
Epoch17/19
17Train Loss:0.0337  Train Acc:0.9903
17Val Loss:0.1660  Val Acc:0.9596
----------
Epoch18/19
18Train Loss:0.0309  Train Acc:0.9904
18Val Loss:0.1610  Val Acc:0.9584
----------
Epoch19/19
19Train Loss:0.0334  Train Acc:0.9903
19Val Loss:0.1638  Val Acc:0.9596
```



验证：

```python
from sklearn.metrics import accuracy_score
lstmmodel.eval()
test_y_all = torch.LongTensor()
pre_lab_all = torch.LongTensor()
for step,batch in enumerate(test_iter):
    textdata,target = batch.title[0],batch.label.view(-1)
    out = lstmmodel(textdata)
    pre_lab = torch.argmax(out,1)
    test_y_all=torch.cat((test_y_all,target))
    pre_lab_all=torch.cat((pre_lab_all,pre_lab))
acc = accuracy_score(test_y_all,pre_lab_all)
print("在测试集上的预测精度为：",acc)
precision_recall_fscore_support(test_y_all,pre_lab_all,average='macro')
```

```
在测试集上的预测精度为： 0.903559806725175

(0.8303259137365389, 0.7477799965992963, 0.7800669607416153, None)
```

```python
from sklearn.metrics import roc_auc_score
print ("AUC is",roc_auc_score(test_y_all, pre_lab_all))
```

```
AUC is 0.7477799965992963
```

```python
precision_recall_fscore_support(test_y_all,pre_lab_all,average='weighted')
```

```
(0.8958538591273183, 0.903559806725175, 0.8967018015922545, None)
```

```python
from sklearn.metrics import roc_auc_score
print ("AUC is",roc_auc_score(test_y_all, pre_lab_all))
```

```
AUC is 0.7477799965992963
```

5.实际的检验

```python
for step,batch in enumerate(test_iter):
    textdata,target = batch.title[0],batch.label.view(-1)
    out = lstmmodel(textdata)
    pre_lab = torch.argmax(out,1)
    test_y_all=torch.cat((test_y_all,target))
    pre_lab_all=torch.cat((pre_lab_all,pre_lab))
    print("真正标签：",target)
    print("猜测：",pre_lab)
    break
```

```
真正标签： tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
猜测： tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

五、评价指标的理解

指标：

accuracy：在全部预测中，正确预测结果占的比例

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

precision：全部阳性预测中，正确预测结果占的比例

recall：就是在全部阳性事件中，正确预测结果占的比例

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$

F1: precision 和 recall 的调和平均值

ROC：描述分类器的 True Positive Rate（TPR，分类器分类正确的正样本个数占总正样本个数的比例）与 False Positive Rate（FPR，分类器分类错误的负样本个数占总负样本个数的比例）之间的变化关系。

AUC：ROC 曲线下的面积值

average 参数：

micro：微平均，即先将多个混淆矩阵的 TP,FP,TN,FN 对应的位置求平均，然后按照 PRF 值公式及逆行计算。在单标签分类下 PRF 各个值和 acc 值一样。

macro：宏平均，对多个混淆矩阵求 PRF，然后求 PRF 的算术平均

weighted：针对类别不均衡情况，对于 macro 中每一类的 PRF 给予不同的权重值相加实现

binary：未指定 pos_label 时默认取 1，即将 1 视为阳性，给出 PRF 值

本次试验主要使用 macro 和 weighted

六、实验反思

1.超参，学习率和 epoch 数量的选择。根据验证集调节 lr 和 epoch_num，直至 val 的曲线平滑不发生突然变化为止。

2.词向量化，选择的是 nn 的 embedding 层，如果是使用 word2vec，是否更好一

些

3.样本比例的选取。一般的训练集、验证集、测试集的比例为 8:1:1。而本实验测试集样本量比训练集样本还多

4.样本不均衡，真新闻比假新闻多很多，导致假新闻的特征提取不便。

解决方法：1.过抽样，欠抽样 2.增加权重，不同类别设置不同的权重值 3.调节阈值

5.机器学习和深度学习最后的效果差不多，也许是数据量还不够大所导致，样本还是不够