人工智能学习笔记七——基于 BiLstm 的恶意 url 检测

本文将用 BiLstm 模型，对于恶意的 url 访问进行检测，从而保证网络空间的安全。

首先在介绍 BiLstm 模型之前，先介绍一下 Lstm 长短期记忆神经网络模型。

长短时记忆网络长短时记忆网络（Long Short-Term Memory Network，简称 LSTM）是循环神经网络模型（Recurrent Neural Network，简称 RNN）的一个重要分支，具有 RNN 的优点并在其基础上进行改善。早期的 DFN、CNN、BP 等深度学习网络的输出都只考虑前一个输入的影响，而不考虑其它时刻输入的影响，对于简单的非时间序列和图像的分析有较好的效果，比如单个词语感情分类、乳腺癌检测等。但是，对于一些与时间先后有关的，比如本文研究的连续时间下一个拼音转化等等，只考虑前一时刻输入的网络模型预测表现不佳。在 DFN、CNN 等简单的多层神经网络中，隐藏层之间各个节点分开工作、互不相关。而在 RNN 中，隐藏层各个节点之间会互相影响，各神经单元之间形成连接，呈现出动态时间序列行为。

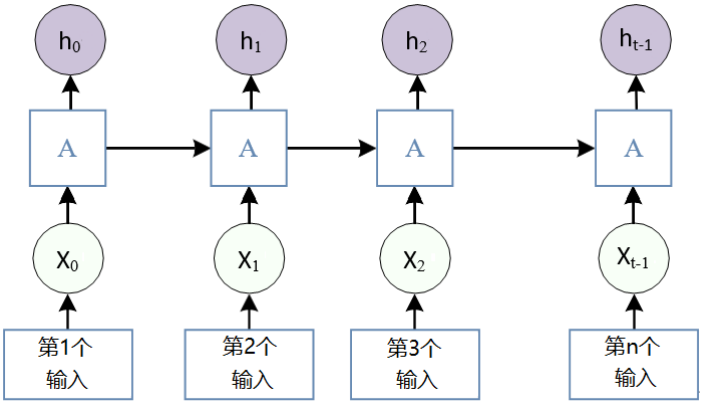

图1 RNN循环神经网络示意图

如图 1 所示，可以把一句句子当中的每一个元素依次输入到 RNN 网络当中，x_i为第 i 个元素的输入，hi 为对应位置的隐藏状态，而h_t与x_t为序列中最后一个单元，y 为模型输出。图中第 i 层的隐藏状态 h，包含前 i 个输入所提供的信息，就是 RNN 网络具有记忆力的原因，其中h_i由 i 时刻的输入x_i与上一时刻的隐藏状态h_{i-1}通过计算得到，可以发现 RNN 网络最终的隐藏状态中将包含对整

个输入序列信息的抽象化表示。h_i和 y 的计算方式分别如下：

$$h_i = F(Ux_i + Wh_{i-1})$$

其中：h_i为i时刻的隐藏层状态；F为激活函数；U为权重矩阵；x_i为i时刻的输入；W为权重矩阵；h_{i-1}为i-1 时刻的隐藏层状态。

$$y = G(Vh_i)$$

其中：y为模型输出；G为激活函数；V为权重矩阵；h_i为第 i 时刻的隐藏层状态。

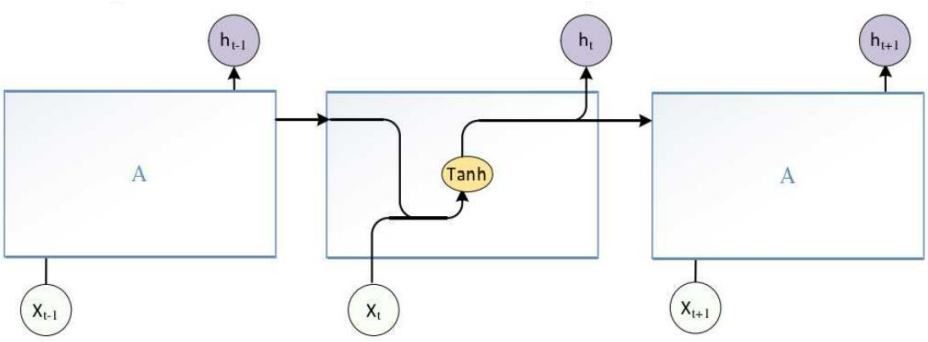

图 2 RNN 最小单元模型

循环神经网络 RNN 理论上可以处理任意长度的时间序列，但实际上，标准 RNN 模型在处理时间跨度较长的时序过程中，随着模型信息传递，最早的信息会失去效力，RNN 无法建立远程结构连接，存在梯度消失或称梯度爆炸问题，可以通过设置超参数、修改激活函数、Dropout 剪枝等方式改善。但，这些方法无法从根本上解决 RNN 梯度消失问题因而精度提升效果较差，为解决时序长时依赖性问题，长短时记忆网络 LSTM 被提出。

LSTM 神经元结构如图 3 所示。其中x_t为 t 时刻输入，h_t为 t 时刻隐藏层状态，C_t为 t 时刻单元内部状态，σ与 tanh 为激活函数。

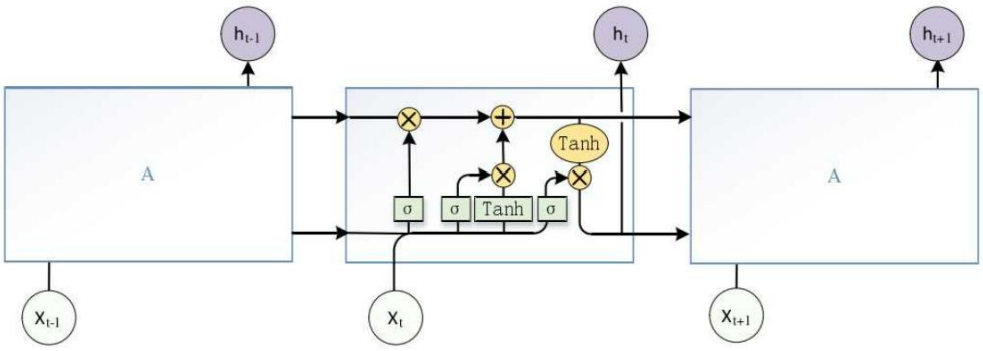

图 3 LSTM 最小模型单元

LSTM 与 RNN 一样，也是通过内部状态的传递来发掘序列元素间的依赖关

系。LSTM 为了解决 RNN 梯度更新上的缺陷，引入了门控机制，门控由激活函数神经层和逐点乘法运算组成，可以有选择的让信息通过。LSTM 的门控环节分为遗忘、输入、输出三部分并且引入了一个状态单元协调整个网络的运作。

（1） 遗忘门

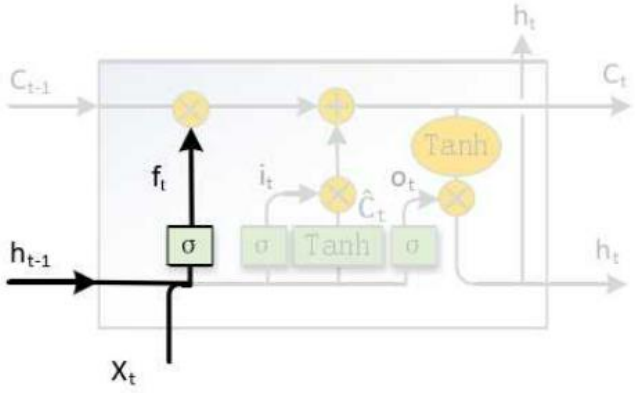

图 4 LSTM 遗忘门

LSTM 遗忘门如图 4 所示，遗忘门的作用是来决定对上一时刻传入信息的保留程度。遗忘门f_t是将t时刻的输入xt与t − 1 时刻隐藏层输出h_{t-1}通过线性变换，再施加激活函数σ得到的，计算方法如下。

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

其中：f_t为遗忘门；σ为激活函数；W_f为遗忘门参数；x_t为t时刻的输入；U_f为遗忘门参数；h_{t-1}为$t-1$时刻隐藏层输出；b_f为遗忘门参数。

（2） 输入门

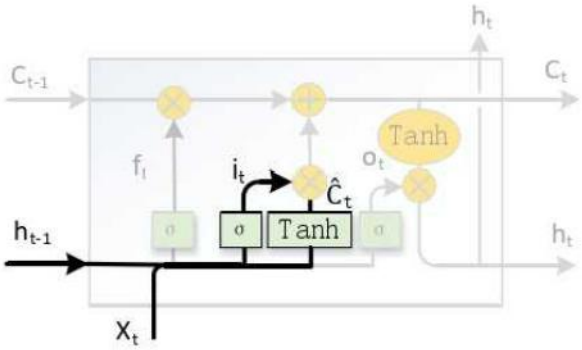

图 5 LSTM 输入门

LSTM 输入门如图 5 所示，输入门的工作主要决定t时刻输入信息的保留程

度。输入门 i_t的计算方式与遗忘门f_t相似，如下式所示。

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

其中：i_t为输入门；σ为激活函数；W_i为输入门参数；x_t为t时刻的输入；U_i为输入门参数；h_{t-1}为$t-1$时刻隐藏层输出；b_i为输入门参数。$\tilde{C}_t$用来描述t时刻输入状态，由$t-1$时刻隐藏层输出h_{t-1}与 t 时刻的输入x_t,经由线性变换再施加 tanh 求得，如下式所示，$\tilde{C}_t$相当于将输入x_t，$t-1$时刻隐藏层状态h_{t-1}所包含的状态信息进行了整合，形成了一个新的状态量。

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

其中：$\tilde{C}_t$为t时刻输入状态；tanh 为激活函数；W_c为输入状态参数；x_t为 t 时刻的输入；U_c为输入状态参数；h_{t-1}为$t-1$时刻隐藏层状态b_c为输入状态参数。

（3） 单元状态

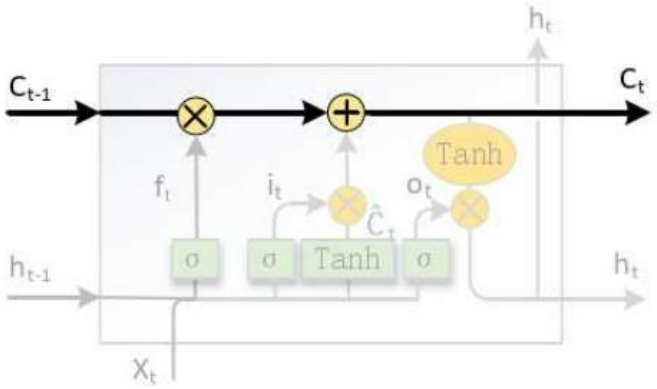

图 6 LSTM 单元状态

LSTM 的单元状态，如图 6 所示。单元状态是贯穿整个 LSTM 网络的信息传送带，让文本信息以不变的方向流动。

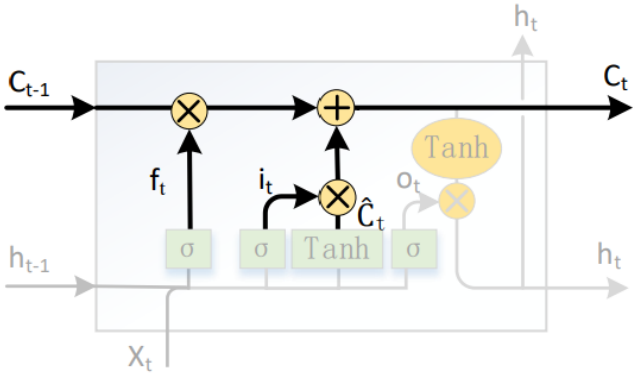

图 7 LSTM 单元状态更新

状态单元更新如图 6 所示，其主要作用是更新 LSTM 的内部状态，将上一时刻的内部状态C_{t-1}更新为该时刻的内部状态C_t。C_t的计算公式如下式所示，首先通过C_{t-1} · f_t的方式来决定$t-1$ 时刻信息的保留度，然后将保留下来的信息$\tilde{C}_t \cdot i_t$与该时刻状态信息相加，计算出t时刻的输出C_t。

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

其中：C_t为t时刻内部状态；C_{t-1}为$t-1$ 时刻内部状态；f_t为遗忘门；$\tilde{C}_t$为t时刻输入状态；i_t为输入门。

（4） 输出门

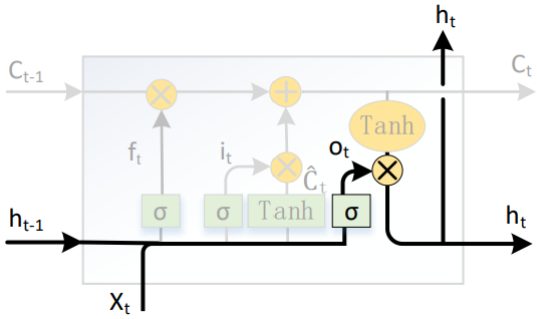

图 8 LSTM 输出门

LSTM 输出门如图 8 所示，输出门控制t时刻的输出取决于状态单元C_t的程度。输出门 o_t方式也与f_t、i_t类似，如下式所示。

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

其中：o_t为输出门；σ为激活函数；W_o为输出门参数；x_t为t时刻的输入；U_o为输出门参数；h_t-1 为$t-1$ 时刻隐藏层输出；b_o为输出门参数。最终 t 时刻的隐藏状态输出 h_t，是由t时刻的内部状态C_t与输出门o_t共同决定，其计算公式如下式所示。

$$h_t = o_t \cdot Tanh(c_t)$$

其中：h_t为t时刻隐藏层输出；o_t 为输出门；Tanh 为激活函数；c_t 为t时刻内部状态

但在标准的 LSTM 中，单元状态的传输是从前往后单向的，所以 LSTM 模型只能学习过去时刻的文本特征而无法学习未来时刻的文本特征。而双向长短期记忆网络（Bidirectional Long Short-Term Memory，简称 BiLSTM）有两条单元状态传送带，分别传递从前往后和从后往前的信息，使得 BiLSTM 模型能够

在利用过去时刻的文本数据信息的同时，能够学习未来时刻文本信息的特征，并对其进行递归和反馈，预测结果比单向 LSTM 更加准确，挖掘时间序列过去和未来数据的联系，提高数据利用率更好利用时序的时间特征，以提高模型预测准确度。 BiLSTM 网络结构如图 9 所示，BiLSTM 网络是正向和反向结合的双向循环结构，可以更好的挖掘时序数据的关联特征。

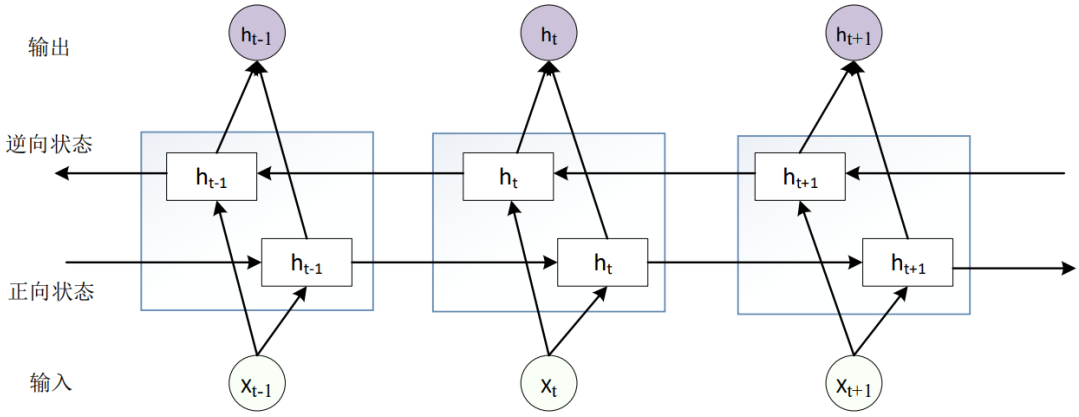

图 9 BiLSTM 网络结构

假设$\overrightarrow{h_t}$为t时刻正向 LSTM 网络的隐藏层状态，其计算公式如下所示。可以看作是单层的 LSTM 网络，由$t-1$时刻状态$\overrightarrow{h_{t-1}}$，计算t时刻状态$\overrightarrow{h_t}$ 的过程，x_t为t时刻的输入。

$$\overrightarrow{h_t} = LSTM(x_t, \overrightarrow{h_{t-1}})$$

其中：$\overrightarrow{h_t}$为t时刻正向 LSTM 网络的隐藏层状态；LSTM 为 LSTM 单元；x_t为t时刻的输入；$\overrightarrow{h_{t-1}}$为$t-1$时刻状态正向 LSTM 网络的隐藏层状态。类似的$\overrightarrow{h_t}$为t时刻反向 LSTM 网络的隐藏层状态，则其计算式如下式所示：

$$\overleftarrow{h_t} = LSTM(x_t, \overleftarrow{h_{t-1}})$$

其中：$\overleftarrow{h_t}$为t时刻正向 LSTM 网络的隐藏层状态；LSTM 为 LSTM 单元；xt为t时刻的输入；$\overleftarrow{h_{t-1}}$为$t-1$时刻状态正向 LSTM 网络的隐藏层状态。BiLSTM 网络输出就是两部分隐藏 层状态$\overleftarrow{h_t}$与$\overrightarrow{h_t}$组合在一起，从而构成网络整体隐藏状态 h_t。

接下来介绍一下数据集，数据集摘自 mirrors / Echo-Ws / UrlDetect · GitCode，正例是某网站服务器上一天正常的访问 url，负例是收集了网上以及其他 github

仓库的大佬提供的数据，包括了网络攻击中常见的 sql 注入攻击，xss 攻击等。在这里，笔者只用到了 badqueries.txt 做为负例，good_fromE2.txt 做为正例。

有了训练数据集，首先需要进行数据清洗，观察了一下 badqueries.txt 这个数据集，发现其中一些数据甚至都不是 url 链接，因而需要剔除掉。接着，就是分割数据，我们使用 jieba 分词，将每个 url 请求分割成元素数组，便于后续的训练。代码如下：

```
import jieba
fr = open("badqueries.txt",encoding="utf-8")
databad=[]
for i in fr.readlines():
    if i[0]=="/" and i[-1]!="/":
        databad.append(list(jieba.cut(i)))
fr.close()
datagood=[]
fr = open("good_fromE2.txt",encoding="utf-8")
for i in fr.readlines():
    if i[0]=="/" and i[-1]!="/":
        datagood.append(list(jieba.cut(i)))
fr.close()
#print(len(datagood))#21911
#print(len(databad))#32014
```

发现正例一共有 21911 条数据，负例一共有 32014 条数据。

由于计算机只能对于数字进行计算，不能对字符运算，因而接下来需要给数据里的每个元素添加一个标签，将所有元素转化成数字。但又由于数据集的数字过多，元素的数量过于庞大，如果每一个元素都标注标签，反而会加大计算机的运算量。

图 9 元素标签标注图

图 9 是按照元素出现次数排列过后的标签标注图，由图可知，里面有不少的元素是数据集的提供者自定义的名称，并没有实际含义，所以需要删掉，因而这边标注标签的分词器只选取了出现频率次数最高的 800 个元素进行标签，其余的数据大概率为自定义的名称，因而进行剔除操作。代码如下：

```
from keras.preprocessing.text import Tokenizer
numchar=800
tokenizer_str = Tokenizer(num_words=numchar)
tokenizer_str.fit_on_texts(databad+datagood)
#print("word_index: \n",tokenizer_str.word_index)
```

接着对数据进行对齐操作，笔者把所有超过 30 元素的数据删除掉，所有不到 30 个元素的数据进行补零操作，这样确保所有的数据都是 30 长度的，从而确保能够代入模型当中进行训练。然后制作输出样本，规定正例的输出为 1，负例的输出为 0，代码如下：

```
from keras_preprocessing.sequence import pad_sequences
import numpy as np
datagood_ls=[]
databad_ls=[]
length = 30#对长度超过 30 的数组进行删除
for i in datagood:
    if len(i)<length:
        out = tokenizer_str.texts_to_sequences([i])
        datagood_ls.append(out[0])
for i in databad:
```

```
    if len(i)<length:
        out = tokenizer_str.texts_to_sequences([i])
        databad_ls.append(out[0])

dataall = datagood_ls + databad_ls
dataout = [1]*len(datagood_ls) + [0]*len(databad_ls)
# 把样本都 pad 到 30 个词，转化成 numpy 数组
data_all_mat = pad_sequences(dataall, maxlen=length, padding='post')
dataout = np.array(dataout)
#print(len(data_all_mat))#47458
#print(len(dataout))
```

接着就可以训练了，这边先将输入数据进行 word embedding 操作，将数据变成 256 维的数组，然后搭建 3 层 BiLstm 模型，对最后一层模型的输出进行 sigmoid 操作，如图 10 所示。

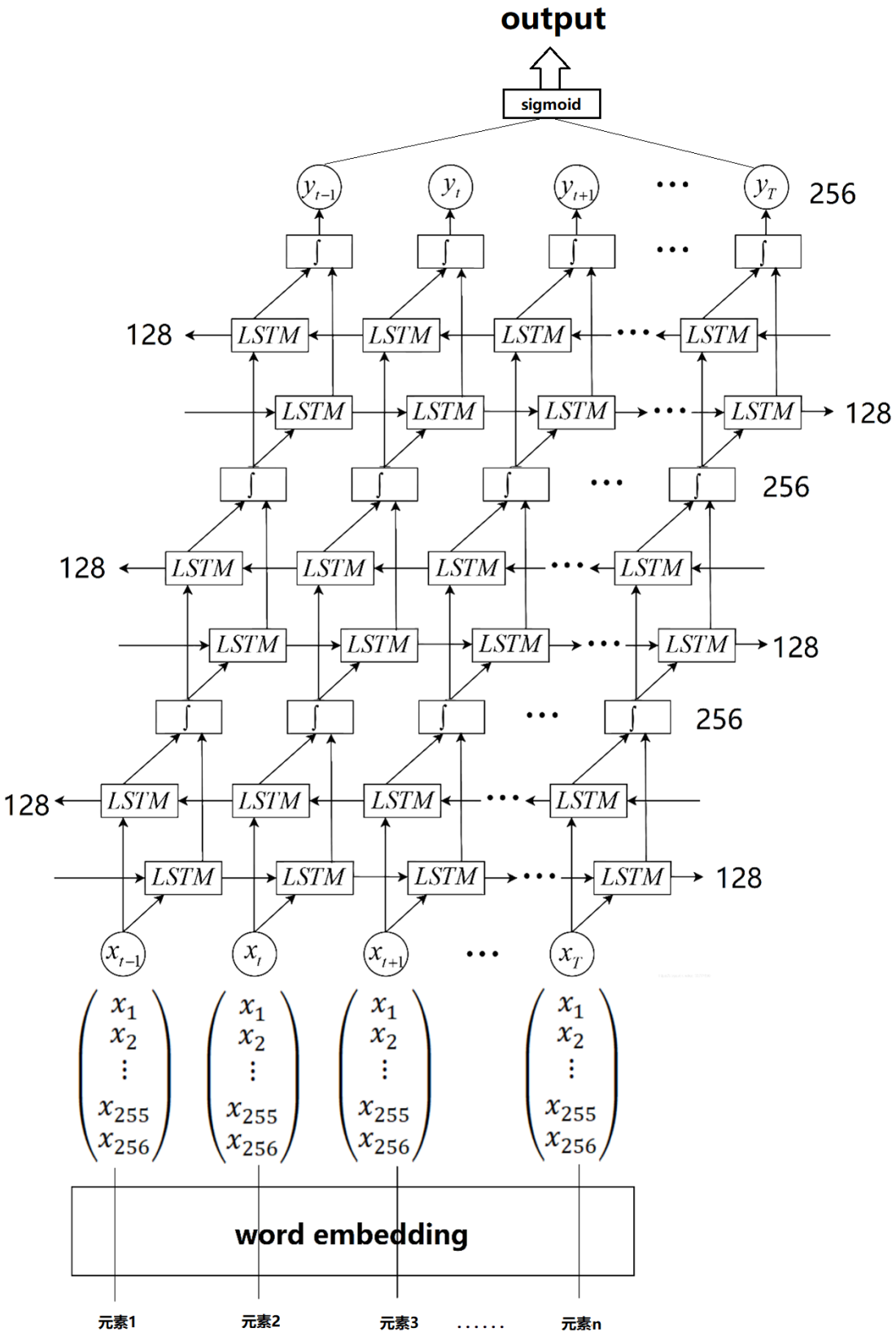

图 10 网络示意图

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 30, 256)           204800

 bidirectional (Bidirectiona  (None, 30, 256)           394240
 l)

 dropout (Dropout)            (None, 30, 256)           0

 bidirectional_1 (Bidirectio  (None, 30, 256)           394240
 nal)

 dropout_1 (Dropout)          (None, 30, 256)           0

 bidirectional_2 (Bidirectio  (None, 256)               394240
 nal)

 dropout_2 (Dropout)          (None, 256)               0

 dense (Dense)                (None, 1)                 257

 activation (Activation)      (None, 1)                 0

=================================================================
Total params: 1,387,777
Trainable params: 1,387,777
Non-trainable params: 0
_________________________________________________________________
```

图 11 网络示意图

训练代码如下：

```
from keras import Sequential
from keras.layers import Embedding
from keras.models import Model,load_model
from keras.layers import Conv2D, MaxPooling2D, Dropout, Activation, Flatten, Dens
e,BatchNormalization,LSTM,Bidirectional
import pickle
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
data_train, data_test, dataout_train, dataout_test = train_test_split(data_all_ma
t, dataout, test_size=0.3, random_state=42, shuffle=True)
model = Sequential()
model.add(Embedding(numchar, 256, input_length=length))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(128, return_sequences=False)))
model.add(Dropout(0.5))
```

```
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
history = model.fit(data_train,dataout_train, epochs=5, batch_size=128, validatio
n_data=(data_test, dataout_test),verbose=2)
# 保存模型
model.save('all.h5')

# 保存分词器

with open('dataall.pickle', 'wb') as handle:
    pickle.dump(tokenizer_str, handle, protocol=pickle.HIGHEST_PROTOCOL)


plt.figure()
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
# 绘制训练 & 验证的损失值
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

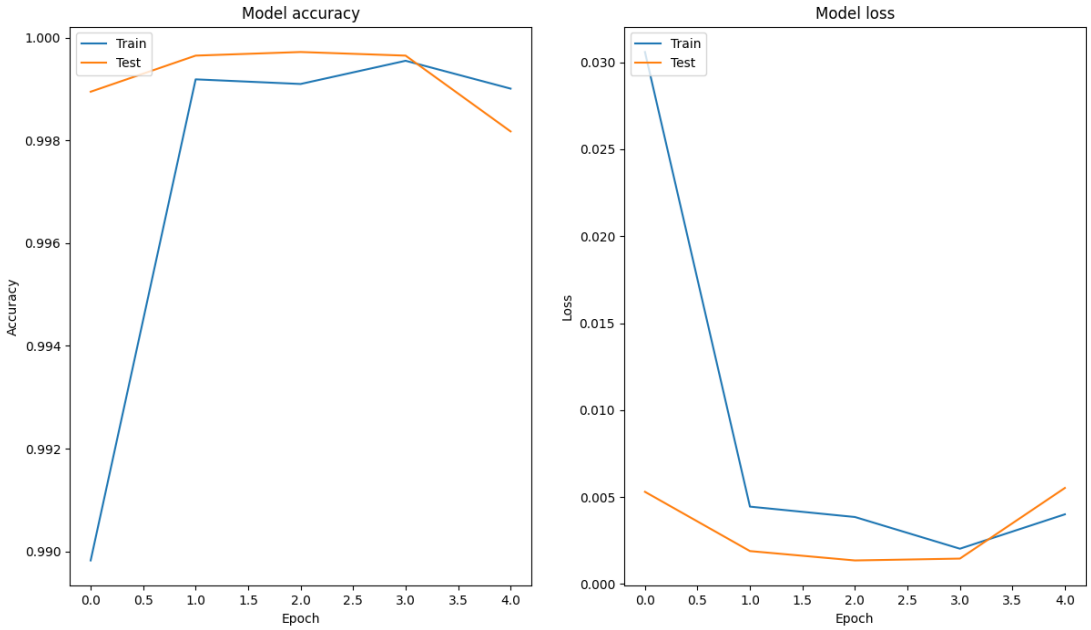

图 12 训练过程图

```
Epoch 1/5
260/260 - 137s - loss: 0.0306 - accuracy: 0.9898 - val_loss: 0.0053 - val_accuracy: 0.9989 - 137s/epoch - 529ms/step
Epoch 2/5
260/260 - 121s - loss: 0.0044 - accuracy: 0.9992 - val_loss: 0.0019 - val_accuracy: 0.9996 - 121s/epoch - 467ms/step
Epoch 3/5
260/260 - 207s - loss: 0.0039 - accuracy: 0.9991 - val_loss: 0.0014 - val_accuracy: 0.9997 - 207s/epoch - 797ms/step
Epoch 4/5
260/260 - 219s - loss: 0.0020 - accuracy: 0.9995 - val_loss: 0.0015 - val_accuracy: 0.9996 - 219s/epoch - 840ms/step
Epoch 5/5
260/260 - 232s - loss: 0.0040 - accuracy: 0.9990 - val_loss: 0.0055 - val_accuracy: 0.9982 - 232s/epoch - 893ms/step
```

图 13 训练过程图

为了展示一下模型的效果，笔者这边选了自定义的 2 个正例，6 个负例的 url 输入到模型进行预测，正例分别是笔者个人的网站链接和上海理工大学的校园网主页链接，负例是 ctf 中关于网络攻击的题型的恶意 url 攻击。代码如下：

```
import pickle
from keras.models import Model,load_model
import jieba
from keras_preprocessing.sequence import pad_sequences
with open('dataall.pickle', 'rb') as f:
    tokenizer_string = pickle.load(f)
model=load_model("all.h5")


length=30
test_string_set=["https://www.usst.edu.cn/main.htm",
                 "https://caodong0225.github.io/index1.html",
                 "http://challenge-
f116bd91836c903a.sandbox.ctfhub.com:10800/www.zip",
                 "http://challenge-405b4e1c032f6e09.sandbox.ctfhub.com:10800/?id=-
1+union+select+1%2C%28select+flag+from+sqli.flag+limit+0%2C1%29",
                 "http://8ec64a9a-35a9-4d04-9919-
ca514a9ff904.node4.buuoj.cn:81/level2?username=<script>alert('xss')</script>",
```

```
                "http://challenge-
73c6b2c2f2993e54.sandbox.ctfhub.com:10800/?url=http://127.0.0.1/flag.php",
                "http://challenge-
be3e0224b7f532fa.sandbox.ctfhub.com:10800/?url=file:///var/www/html/flag.php",
                "http://challenge-
1511574cb1da7fe5.sandbox.ctfhub.com:10800/?url=http://notfound.ctfhub.com@127.0.0.1
/flag.php"]

for test_string in test_string_set:
    test_string_token = tokenizer_string.texts_to_sequences([list(jieba.cut(test_st
ring))])
    test_string_mat = pad_sequences(test_string_token, maxlen=length, padding='post
')
    print(test_string)
    print(model.predict(test_string_mat,verbose=0))
```

```
https://www.usst.edu.cn/main.htm
[[0.99553275]]
https://caodong0225.github.io/index1.html
[[0.88632864]]
http://challenge-f116bd91836c903a.sandbox.ctfhub.com:10800/www.zip
[[0.4560371]]
http://challenge-405b4e1c032f6e09.sandbox.ctfhub.com:10800/?id=-1+union+select+1%2C%28select+flag+from+sqli.flag+limit+0%2C1%29
[[0.3639168]]
http://8ec64a9a-35a9-4d04-9919-ca514a9ff904.node4.buuoj.cn:81/level2?username=<script>alert('xss')</script>
[[0.00896045]]
http://challenge-73c6b2c2f2993e54.sandbox.ctfhub.com:10800/?url=http://127.0.0.1/flag.php
[[3.7267684e-05]]
http://challenge-be3e0224b7f532fa.sandbox.ctfhub.com:10800/?url=file:///var/www/html/flag.php
[[2.855397e-05]]
http://challenge-1511574cb1da7fe5.sandbox.ctfhub.com:10800/?url=http://notfound.ctfhub.com@127.0.0.1/flag.php
[[0.00196837]]
```

图 14 预测输出图

可以看到，结果还是非常可观的。