Configuration Manual

MSc Research Project

Enhancing Malware Detection Using Stacked BiLSTM with Attention Mechanism: A Deep Learning Approach

Srinivasan Masilamani
Student ID: x21159904

School of Computing
National College of Ireland

Supervisor: Arghir Nicolae Moldovan

National College of Ireland

MSc Project Submission Sheet

School of Computing

StudentName:	Srinivasan Masilamani
Student ID:	x21159904
Programme:	Msc Cyber Security **Year:** 2023-2024
Module:	Msc Academic Internship
Supervisor:	Arghir Nicolae Moldovan
Submission Due Date:	25 April 2024
Project Title:	Enhancing Malware Detection Using Stacked BiLSTM with Attention Mechanism: A Deep Learning Approach
WordCount:...912	**Page Count : 10**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Srinivasan Masilamani……

Date: ……25/05/2024 ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

This report proposes a malware detection machine learning approach with Stacked-BiLSTM incorporating Attention Mechanism. In this project we have used some of the common software's that are usually used for machine learning projects. We have implemented four ML models namely CNN, LSTM, BiLSTM and Stacked-BiLSTM with Attentions mechanism. Where we compare the results based on Accuracy and confusion matrix to understand which model outperforms and will be best to further implement in real world applications. Firstly, whenever we do a ML classification problem, we need to structure our methodology according to the dataset in order to yield good and accurate results. So we need to first get the dataset, upload and clean the dataset ie; preprocess, Select the features of interest in the problem, Split the data for training and evaluation, Select and design a Model accordingly and Train the Model with our processed dataset.
We will see the step-by-step implementation below.

2. Requirements:

This project requires a smooth-running computer with good latest hardware as follows:

- Processor: Intel Core i5/i7 CPU @2.20GHz
- RAM – 8 (or) 16 and above DDR4
- GPU – Onboard / Nvidia GeFoce GTX
- Storage – 128Gb Storage
- OS – Windows 10 and Above / Mac OS

We have used the following Software's to run our Machine Learning Project:

Software	Version
Python	3.8.3
Anaconda Navigator	2.5.2
Jupyter Notebook	7.0.8
Tensorflow	2.16.1
Numpy	1.26.4
Scikit-Learn	1.2.2

Above are some of the packages that we used inside Anaconda Navigator. Anaconda Environment File attached to our project will help create the same environment which we let us stay in the same page in terms of dependencies. (Research_Project_Anaconda_Environment.yaml).
Steps : open Anaconda navigator >> Environment >> Import >> select the yaml file >> Click Create.

3. Initial Setup:

3.1 Importing All Necessary Libraries:

```
import os
import sys
import pickle
import json
import numpy as np
import pandas as pd
import seaborn as sns
import keras.backend as K
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from sklearn import preprocessing
import plotly.figure_factory as ff
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential, model_from_json
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from sklearn.metrics import confusion_matrix, classification_report , accuracy_score, precision_recall_fscore_support
from tensorflow.keras.layers import Conv1D, Dropout, Flatten, Dense,Bidirectional,LSTM, Activation, Layer
import warnings
warnings.filterwarnings("ignore")
```

```
import sklearn
sklearn.__version__
```

```
'1.2.2'
```

3.2 Data Loading:

We use pandas library to load our dataset

```
dataframe = pd.read_csv('/content/drive/MyDrive/pe_malware_classification/Data/dataset_malwares.csv')
dataframe.head()
```

3.3 Data Cleaning:

We use dataframe.describe() to see visually our dataset. Next, we can use the key ie; the columns as required to construct our new dataframe that will only have the essential columns.

```
dataframe.describe()
```

	e_magic	e_cblp	
count	19611.0	19611.000000	19
mean	23117.0	178.615726	
std	0.0	987.200729	1
min	23117.0	0.000000	
25%	23117.0	144.000000	
50%	23117.0	144.000000	
75%	23117.0	144.000000	
max	23117.0	59448.000000	63

8 rows × 78 columns

Have 78 Columns.

```
#removed unnecessary columns
dataframe = dataframe[['e_magic', 'e_cblp', 'e_cp', 'e_crlc', 'e_cparhdr', 'e_minalloc','e_maxalloc', 'e_ss', 'e_sp', 'e_csum', 'e_ip', 'e_cs', 'e_lfarlc','e_ovno', 'e_oemid', 'e_oeminfo',
       'NumberOfSections', 'TimeDateStamp', 'PointerToSymbolTable','NumberOfSymbols', 'SizeOfOptionalHeader', 'Characteristics', 'Magic','MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfC
       'SizeOfInitializedData', 'SizeOfUninitializedData','AddressOfEntryPoint', 'BaseOfCode', 'ImageBase', 'SectionAlignment','FileAlignment', 'MajorOperatingSystemVersion','MinorOperating
       'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfHeaders','CheckSum', 'SizeOfImage', 'Subsystem', 'DllCharacteristics','SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapR
dataframe.head()
```

	e_magic	e_cblp	e_cp	e_crlc	e_cparhdr	e_minalloc	e_maxalloc	e_ss	e_sp	e_csum	...	SizeOfImage	Subsystem	DllCharacteristics	SizeOfStackReserve	SizeOfStackCommit	SizeOfHeapReser
0	23117	144	3	0	4	0	65535	0	184	0	...	274432	2	32832	524288	8192	10485
1	23117	144	3	0	4	0	65535	0	184	0	...	442368	2	33088	1048576	4096	10485
2	23117	144	3	0	4	0	65535	0	184	0	...	49152	2	0	1048576	4096	10485
3	23117	144	3	0	4	0	65535	0	184	0	...	1032192	2	32768	2097152	4096	10485
4	23117	144	3	0	4	0	65535	0	184	0	...	110592	2	0	2097152	4096	10485

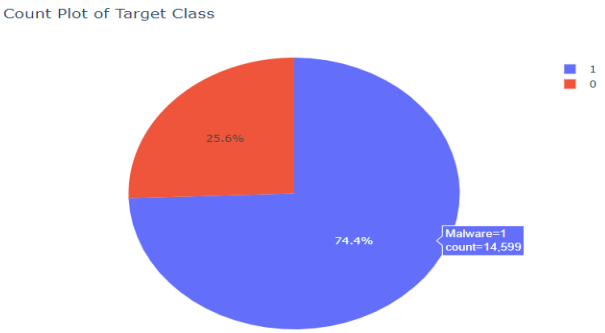
5 rows × 53 columns

Now it has 53 columns, were we chose to remove the unwanted columns by selecting only the necessary columns.

3.4 Data Visualization:

We can further use MatplotLib, Plotly.express etc., and visualize our dataset, for example a pie chart.

```
tempdf = dataframe['Malware'].value_counts().reset_index()
fig = px.pie(tempdf, values='count', names='Malware', color='Malware', title="Count Plot of Target Class")
fig.show()
```



3.5: Data Preprocessing:

To balance the dataset and ensure a fair representation of both classes, we use method SMOTE oversampling are used considering the possible class imbalance of course between benign and malicious samples.

```
#splitting data into feature and target
X = dataframe.drop('Malware',axis=1)
Y = dataframe['Malware']

#Data Balancing using Smote OverSampling Technique (apply this beacuase data was imbalanced)
oversample = SMOTE()
X, Y = oversample.fit_resample(X,Y)
```

3.6 Split Data into Train and Evaluation:

Using the function train_test_split from sklearn.model_selection library we split our data for training and evaluating the model.

```
X_train, X_val, y_train, y_val = train_test_split(X,Y,test_size=0.1)
X_train, X_test, y_train, y_test = train_test_split(X_train,y_train,test_size=0.1)

X_train.shape
```

4. Deep Learning Model:

```
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)

X_train = np.expand_dims(X_train,axis=2)
X_val = np.expand_dims(X_val,axis=2)
X_test = np.expand_dims(X_test,axis=2)

cls = 2
```

As part of Preprocessing for a classification problem we set the “to_categorical” function which is converting the target labels into one-hot encoded vector. The “expan_dims” function is used to add an extra dimension to the input data to match the expected input shape of the model.

4.1 CNN

```
model = Sequential()
model.add(Conv1D(32,2 ,activation='relu', input_shape=(X_test.shape[1],X_test.shape[2])))
model.add(Conv1D(32,2 ,activation='relu'))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(cls, activation='softmax'))
model.compile(optimizer="Adam",loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

We here have defined a sequential model using CNN layers for 1D data. Consisting 2 Conv1D layers, a droupout layer, and two dense layers, a flattening layer, with softmax activation compiled with Adam optimizer using categorical_crossentropy loss.

4.2 LSTM

```
model=Sequential()
model.add(LSTM(8, return_sequences=False, input_shape=(X_test.shape[1],1)))
model.add(Dropout(0.5))
model.add(Dense(8,activation='relu'))
model.add(Dense(cls, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer="Adam",metrics=['accuracy'])
model.summary()
```

We here have defined a sequential model using LSTM, consisting LSTM layer, dropout and two dense layer with ReLu Activaition and a softmax activated dense output layer for classification into two classes, also here complied with Adam optimizer using categorical crossentropy loss.

4.3 BiLSTM

```
model=Sequential()
model.add(Bidirectional(LSTM(8, return_sequences=True), input_shape=(X_train.shape[1],1)))
model.add(Dense(8, activation='relu'))
model.add(Flatten())
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(cls, activation='sigmoid'))
model.compile(loss='categorical_crossentropy',optimizer="rmsprop",metrics=['accuracy'])
model.summary()
```

We have here defined a sequential BiLSTM model, which processes input sequence bidirectionally. Consists of Dense layer with ReLu Activation and dropout regularization, ending with sigmoid activated output layer for labeled classification. Also compiled using RMSprop optimized with categorical crossentropy loss.

4.4 Stacked BiLSTM with Attention Mechanism

4.4.1 Attention mechanism Class

```
class Attention(Layer):
  def __init__(self,**kwargs):
    super(Attention,self).__init__(**kwargs)

  def build(self,input_shape):
    self.W=self.add_weight(name="att_weight",shape=(input_shape[-1],1),initializer="normal")
    self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1),initializer="zeros")
    super(Attention, self).build(input_shape)

  def call(self,x):
    et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
    at=K.softmax(et)
    at=K.expand_dims(at,axis=-1)
    output=x*at
    return K.sum(output,axis=1)

  def compute_output_shape(self,input_shape):
    return (input_shape[0],input_shape[-1])

  def get_config(self):
    return super(Attention,self).get_config()
```

This is our incorporation approach where we have defined a class that implements the attention layer for the neural network. This will enable the model to focus on specific feature in the input. Consists of trainable weights for attention calculation and applies them to the input.

4.4.2 Stacked-BiLSTM with Attention Layer Model

```
model= Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=(X_train.shape[1],1)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Bidirectional(LSTM(64, return_sequences=True)))
model.add(Attention())
model.add(Dense(64,activation='relu'))
model.add(Flatten())
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(cls,activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
model.summary()
```

Now here we have utilized our attention mechanism along with stacked BiLSTM, Consist of multiple layers of BiLSTM cells followed by the Attention Layer. This will help capture the important pattern in the data and attend only the relevant information.

5. Training and Evaluation:

5.1 Training the Model :

```
history = model.fit(X_train,y_train,batch_size=32,epochs=5,verbose=1,validation_data=(X_val, y_val))
```

This line of code trains the defined model, so we can use this to train each model and then visualize the result from the output.

5.2 Visualizing the Result:

5.2.1 Plot:

```
#Train and Validation Accuracy
plt.plot(history.history['accuracy'], 'b', marker='o', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'r', marker='o', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(history.history['loss'], 'b', marker='o', label='Training loss')
plt.plot(history.history['val_loss'], 'r', marker='o', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

This line of code will create a visual plot for both training and validation , Accuracy and Loss respectively.

5.2.2 Confusion matrix:

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred,axis=1)
```

```
83/83 [==============================] - 2s 8ms/step
```

```
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlOrBr" ,fmt='g')
plt.show()
```

The above line of code will create a confusion matrix with which we can understand the TP, FP, TN and FN's.

5.2.3 Classification Reports:

```
#Classification Report
print("Classification Report : ")
print(classification_report(y_test, y_pred))
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.96      0.97      1339
           1       0.96      0.99      0.97      1289

    accuracy                           0.97      2628
   macro avg       0.97      0.97      0.97      2628
weighted avg       0.97      0.97      0.97      2628
```

```
res = []
for l in range(cls):
  prec,recall,_,_ = precision_recall_fscore_support(y_test==l,
  y_pred==l,
  pos_label=True,average=None)
  res.append([l,recall[0],recall[1]])
pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.986036	0.957431
1	1	0.957431	0.986036

Above line of code will help us a tabulate the result that will let us understand the recall, f1score, accuracy, sensitivity, and specificity of the trained model.

5.3 Saving the model:

```
model_json = model.to_json()
with open("/content/drive/MyDrive/pe_malware_classification/Models/bestmodel.json", "w") as json_file:
  json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("/content/drive/MyDrive/pe_malware_classification/Models/bestmodel.h5")
print("Saved model to disk")
```

With the Above line of code we can save the model into a file, and then we can utilize this file for further development for example, create a web application that can detect a malware file.

6. Conclusion:

This Configuration manual will help in implementing our machine learning model along with its output. Where, Stacked-BiLSTM with attention mechanism outperforms the other models discussed above. This method implementation provides efficient way of detecting legitimate PE malware files.