HTCS6706_Assignment_Amir

June 25, 2023

1 HTCS 6706 - Data Analysis Assignment

1.0.1 Important Instructions:

To add text content to a cell, you need to follow the steps below: 1 Click on the cell that you want to add content to. 2 Change the cell type to "Code" by selecting "Code" from the dropdown menu at the top of the Jupyter notebook. 3 Add your response as you would in a normal text editor. 4 Change the cell type back to "Markdown" by selecting "Markdown" from the dropdown menu.

Make sure to save your work regularly to avoid losing any changes.

Complete this cell first:

1.0.2 Student Name: Amirali Sarkhosh

1.0.3 Student ID: 1534759

1.0.4 Student Declaration:

By submitting this assessment, I confirm that 1 This is an original assessment and is entirely my own work. 2 The work I am submitting for this assessment is free of plagiarism.
3 I have read and understood the Academic Integrity Procedure here. 4 I have also read and understood the Student Disciplinary Statue here. 5 Where I have used ideas, tables, diagrams etc of other writers, I have acknowledged the source in every case.

1.0.5 Business Case Overview (200-250 words):

Elaborate on the business case by identifying a cybersecurity-related business gap or need with reference to the literature. Use IEEE referencing style and add your reference list in the last cell of the Jupiter notebook file. Write your response in the following cell [10 Marks].

1. Business Case Overview: In the digital era, cybersecurity has appeared as a sizeable subject for corporations across the globe. The growing sophistication of cyber threats, coupled with the developing reliance on virtual platforms, has created a pressing need for robust cybersecurity measures [1]. A significant business gap lies in detecting and preventing these cyber threats, particularly those that breach network security. The KDDTrain+ dataset, a benchmark dataset in the field of intrusion detection, offers a potential solution to this business gap. This dataset, derived from the 1998 DARPA Intrusion Detection Evaluation Program conducted by MIT Lincoln Lab, is widely recognised for its comprehensive range of features that describe different aspects of network connections [2]. These features provide a granular view of network traffic, enabling the identification of patterns and anomalies that may indicate a cyber threat. By leveraging the KDDTrain+ dataset, businesses can enhance their

Intrusion Detection Systems (IDS), making them more effective in detecting and preventing cyber threats. This could lead to developing more targeted and effective cybersecurity measures, thereby addressing the identified business gap. The potential insights from this analysis could inform the design of more sophisticated IDS, contributing to the broader field of cybersecurity.

1.0.6 Dataset Overview (200-250 words):

Provide a list of data sources along with respective links/URLs. Please ensure that attached compressed archives of the respective data do not exceed 50 MB per file. Describe the dataset and its important variables that are relevant to the business case you identified above. [10 Marks]

2. Dataset Overview: The KDDTrain dataset [3], a derivative of the 1998 DARPA Intrusion Detection Evaluation Program, is a comprehensive dataset used for network intrusion detection. It is accessible for download at the Canadian Cybersecurity Agency website or from Kaggle [4]. This dataset is a revised model of the KDD'99 dataset, which addresses some of the inherent issues of the original dataset, along with redundant records and bias in the direction of extra frequent records. The KDDTrain+ dataset includes a variety of features that describe different aspects of network connections, such as duration, protocol type, service, flag, src_bytes, dst_bytes, and many others. These features provide a detailed view of network traffic, enabling the identification of patterns and anomalies that may indicate a cyber threat. In the context of our business case, the KDDTrain+ dataset is highly relevant. The functions that it provides can be used to enhance Intrusion Detection Systems (IDS), making them extra effective in detecting and stopping cyber threats. For instance, the 'duration' characteristic can assist in becoming aware of strangely long connections that might imply a data breach, at the same time as the 'protocol type' and 'service' functions can help locate unauthorised get entry to try. By analysing those variables, enterprises can develop extra focused and powerful cybersecurity measures, addressing the identified gaps in the detection and prevention of cyber threats.

1.0.7 Rationale (200-250 words):

It is essential to show the link between the problem description and key variables in the dataset. Clearly mention the link between these variables and your chosen business case. [10 Marks]

3. Rationale: The business case revolves around enhancing cybersecurity measures by means of improving the detection and prevention of cyber threats. The KDDTrain dataset is helpful in this regard because it presents a variety of information on network connections that can be analysed to detect anomalies and capability threats. The key variables within the dataset that are mainly applicable to this business case include 'duration', 'protocol_type', 'service', 'flag', 'src_bytes', and 'dst_bytes'. These variables offer insights into the nature of network connections, with their duration, the kind of protocol used, the network link at the destination spot, the status of the connection, and the range of data bytes from source to destination. For example, strangely long connection durations ('duration') can identify a data breach, while specific sorts of protocols ('protocol_type') and services ('service') might be related to greater chances of cyber threats. Also, a wide variety of data bytes being despatched from source to destination ('src_bytes' and 'dst_bytes') can propose a possible data leak. By examining these variables, we can monitor patterns and anomalies that would imply a cyber risk. This information can then be utilised to improve Intrusion Detection Systems (IDS), making them

greater powerful in detecting and preventing such risks and threats. Therefore, there is a relationship between those key variables and the enterprise case for developing cybersecurity measures.

1.0.8 Potentials (200-250 words):

Discuss potential insights, outcomes, and products that can emerge from a that analysis that you will introduce in later stages. [10 Marks]

4. Potential Insights The analysis of the KDDTrain+ dataset in the context of the associated business case can yield a wealth of potential insights and outcomes that can extensively improve cybersecurity measures. One potential insight is the identification of patterns and anomalies in network connections that could identify a cyber threat. For instance, the analysis might monitor that some types of protocols or services are greater regularly related to cyber threats. This information might be used to put in force more protection controls for the protocols or offerings, thereby reducing the danger of cyber threats. Moreover, the analysis needs to recognise trends within the size and duration of network traffic that might be determined as a data breach or a denial-of-service (DoS) attack. These insights may want to inform the development of more advanced Intrusion Detection Systems (IDS), so that could discover the threats more efficiently. In terms of outcomes, the evaluation and analysis should lead to the improvement of extra-focused and powerful cybersecurity measures. For instance, enterprises should use the insights obtained from the evaluations to develop the IDS, making them more capable of detecting and stopping cyber threats. As for products, the analysis could inform the design of new cybersecurity tools or services. For example, a new IDS could be developed that leverages machine learning algorithms to analyse network traffic and detect cyber threats based on the patterns identified in the KDDTrain+ dataset. Such a product could considerably increase an enterprise's cybersecurity strength, addressing the identified business gap.

1.0.9 Descriptive Analysis (Codes):

In the following cells, add all the codes used for the dataset import, data cleaning and descriptive data analysis. The codes should be error-free and return outputs without any issues. For this analysis, choose appropriate variables from your dataset while considering the business problem you introduced earlier. [15 Marks]

[23]:
```
import warnings
import cvxpy as cp
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn import metrics
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
```

```
from tensorflow.keras import regularizers
from tensorflow.keras import initializers
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
pd.set_option('display.max_columns',None)
warnings.filterwarnings('ignore')
%matplotlib inline
```

[50]:
```
data_train = pd.read_csv("KDDTrain+.txt")
print(data_train.shape)
```

```
(125972, 43)
```

[25]:
```
data_train.head()
```

[25]:

	0	tcp	ftp_data	SF	491	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	\
0	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	
1	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	
2	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	
3	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	
4	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	

	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.18	2	2.1	0.00	0.00.1	\
0	0	0	0	0	0	0	0	0	13	1	0.0	0.0	
1	0	0	0	0	0	0	0	0	123	6	1.0	1.0	
2	0	0	0	0	0	0	0	0	5	5	0.2	0.2	
3	0	0	0	0	0	0	0	0	30	32	0.0	0.0	
4	0	0	0	0	0	0	0	0	121	19	0.0	0.0	

	0.00.2	0.00.3	1.00	0.00.4	0.00.5	150	25	0.17	0.03	0.17.1	0.00.6	\
0	0.0	0.0	0.08	0.15	0.00	255	1	0.00	0.60	0.88	0.00	
1	0.0	0.0	0.05	0.07	0.00	255	26	0.10	0.05	0.00	0.00	
2	0.0	0.0	1.00	0.00	0.00	30	255	1.00	0.00	0.03	0.04	
3	0.0	0.0	1.00	0.00	0.09	255	255	1.00	0.00	0.00	0.00	
4	1.0	1.0	0.16	0.06	0.00	255	19	0.07	0.07	0.00	0.00	

	0.00.7	0.00.8	0.05	0.00.9	normal	20
0	0.00	0.00	0.0	0.00	normal	15
1	1.00	1.00	0.0	0.00	neptune	19
2	0.03	0.01	0.0	0.01	normal	21
3	0.00	0.00	0.0	0.00	normal	21
4	0.00	0.00	1.0	1.00	neptune	21

[26]:
```
# Create a list 'columns' containing names for each column of the 'data_train'␣
↪DataFrame.
columns =␣
↪(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragmen
```

```
␣
↪'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','n
␣
↪'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count
␣
↪'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff
␣
↪'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_s
␣
↪'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','outcome','lev
# Assign these column names to the 'data_train' DataFrame.
data_train.columns =
```

[27]:
```
data_train.head()
```

[27]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land
0	0	udp	other	SF	146	0	0
1	0	tcp	private	S0	0	0	0
2	0	tcp	http	SF	232	8153	0
3	0	tcp	http	SF	199	420	0
4	0	tcp	private	REJ	0	0	0

	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	0

	root_shell	su_attempted	num_root	num_file_creations	num_shells
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

	num_access_files	num_outbound_cmds	is_host_login	is_guest_login	count
0	0	0	0	0	13
1	0	0	0	0	123
2	0	0	0	0	5
3	0	0	0	0	30
4	0	0	0	0	121

	srv_count	serror_rate	srv_serror_rate	rerror_rate	srv_rerror_rate
0	1	0.0	0.0	0.0	0.0
1	6	1.0	1.0	0.0	0.0
2	5	0.2	0.2	0.0	0.0

```
3         32          0.0              0.0          0.0              0.0
4         19          0.0              0.0          1.0              1.0

   same_srv_rate  diff_srv_rate  srv_diff_host_rate  dst_host_count  \
0           0.08           0.15                0.00             255
1           0.05           0.07                0.00             255
2           1.00           0.00                0.00              30
3           1.00           0.00                0.09             255
4           0.16           0.06                0.00             255

   dst_host_srv_count  dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                   1                    0.00                    0.60
1                  26                    0.10                    0.05
2                 255                    1.00                    0.00
3                 255                    1.00                    0.00
4                  19                    0.07                    0.07

   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         0.88                         0.00
1                         0.00                         0.00
2                         0.03                         0.04
3                         0.00                         0.00
4                         0.00                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate  dst_host_rerror_rate  \
0                  0.00                      0.00                   0.0
1                  1.00                      1.00                   0.0
2                  0.03                      0.01                   0.0
3                  0.00                      0.00                   0.0
4                  0.00                      0.00                   1.0

   dst_host_srv_rerror_rate  outcome  level
0                      0.00   normal     15
1                      0.00  neptune     19
2                      0.01   normal     21
3                      0.00   normal     21
4                      1.00  neptune     21
```

[28]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125972 entries, 0 to 125971
Data columns (total 43 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   duration                     125972 non-null  int64
 1   protocol_type                125972 non-null  object
 2   service                      125972 non-null  object
```

```
 3   flag                         125972 non-null  object
 4   src_bytes                    125972 non-null  int64
 5   dst_bytes                    125972 non-null  int64
 6   land                         125972 non-null  int64
 7   wrong_fragment               125972 non-null  int64
 8   urgent                       125972 non-null  int64
 9   hot                          125972 non-null  int64
 10  num_failed_logins            125972 non-null  int64
 11  logged_in                    125972 non-null  int64
 12  num_compromised              125972 non-null  int64
 13  root_shell                   125972 non-null  int64
 14  su_attempted                 125972 non-null  int64
 15  num_root                     125972 non-null  int64
 16  num_file_creations           125972 non-null  int64
 17  num_shells                   125972 non-null  int64
 18  num_access_files             125972 non-null  int64
 19  num_outbound_cmds            125972 non-null  int64
 20  is_host_login                125972 non-null  int64
 21  is_guest_login               125972 non-null  int64
 22  count                        125972 non-null  int64
 23  srv_count                    125972 non-null  int64
 24  serror_rate                  125972 non-null  float64
 25  srv_serror_rate              125972 non-null  float64
 26  rerror_rate                  125972 non-null  float64
 27  srv_rerror_rate              125972 non-null  float64
 28  same_srv_rate                125972 non-null  float64
 29  diff_srv_rate                125972 non-null  float64
 30  srv_diff_host_rate           125972 non-null  float64
 31  dst_host_count               125972 non-null  int64
 32  dst_host_srv_count           125972 non-null  int64
 33  dst_host_same_srv_rate       125972 non-null  float64
 34  dst_host_diff_srv_rate       125972 non-null  float64
 35  dst_host_same_src_port_rate  125972 non-null  float64
 36  dst_host_srv_diff_host_rate  125972 non-null  float64
 37  dst_host_serror_rate         125972 non-null  float64
 38  dst_host_srv_serror_rate     125972 non-null  float64
 39  dst_host_rerror_rate         125972 non-null  float64
 40  dst_host_srv_rerror_rate     125972 non-null  float64
 41  outcome                      125972 non-null  object
 42  level                        125972 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 41.3+ MB
```

[29]:
```
# Check for missing or null values
print("Number of missing values in each column:")
print(data_train.isnull().sum())
```

```
Number of missing values in each column:
```

```
duration                        0
protocol_type                   0
service                         0
flag                            0
src_bytes                       0
dst_bytes                       0
land                            0
wrong_fragment                  0
urgent                          0
hot                             0
num_failed_logins               0
logged_in                       0
num_compromised                 0
root_shell                      0
su_attempted                    0
num_root                        0
num_file_creations              0
num_shells                      0
num_access_files                0
num_outbound_cmds               0
is_host_login                   0
is_guest_login                  0
count                           0
srv_count                       0
serror_rate                     0
srv_serror_rate                 0
rerror_rate                     0
srv_rerror_rate                 0
same_srv_rate                   0
diff_srv_rate                   0
srv_diff_host_rate              0
dst_host_count                  0
dst_host_srv_count              0
dst_host_same_srv_rate          0
dst_host_diff_srv_rate          0
dst_host_same_src_port_rate     0
dst_host_srv_diff_host_rate     0
dst_host_serror_rate            0
dst_host_srv_serror_rate        0
dst_host_rerror_rate            0
dst_host_srv_rerror_rate        0
outcome                         0
level                           0
dtype: int64
```

[30]:
```
# Check the types of variables in the DataFrame
print("Types of variables in DataFrame:")
```

```
print(data_train.dtypes)
```

```
Types of variables in DataFrame:
duration                         int64
protocol_type                   object
service                         object
flag                            object
src_bytes                        int64
dst_bytes                        int64
land                             int64
wrong_fragment                   int64
urgent                           int64
hot                              int64
num_failed_logins                int64
logged_in                        int64
num_compromised                  int64
root_shell                       int64
su_attempted                     int64
num_root                         int64
num_file_creations               int64
num_shells                       int64
num_access_files                 int64
num_outbound_cmds                int64
is_host_login                    int64
is_guest_login                   int64
count                            int64
srv_count                        int64
serror_rate                    float64
srv_serror_rate                float64
rerror_rate                    float64
srv_rerror_rate                float64
same_srv_rate                  float64
diff_srv_rate                  float64
srv_diff_host_rate             float64
dst_host_count                   int64
dst_host_srv_count               int64
dst_host_same_srv_rate         float64
dst_host_diff_srv_rate         float64
dst_host_same_src_port_rate    float64
dst_host_srv_diff_host_rate    float64
dst_host_serror_rate           float64
dst_host_srv_serror_rate       float64
dst_host_rerror_rate           float64
dst_host_srv_rerror_rate       float64
outcome                         object
level                            int64
dtype: object
```

[31]:
```
# Calculate descriptive statistics for each column
stats = data_train.describe()
```

[32]:
```
# Stats contains statistical summary,
# calculate the Interquartile Range (IQR) for each column in 'data_train' DataFrame.
# IQR is the difference between the 75th percentile (Q3) and 25th percentile (Q1) of the data.
Q1 = stats.loc['25%']
Q3 = stats.loc['75%']
IQR = Q3 - Q1
# Identify the outliers in the 'data_train' DataFrame.
# Outliers are data points that are below (Q1 - 1.5 * IQR) or above (Q3 + 1.5 * IQR).
outliers = (data_train < (Q1 - 1.5 * IQR)) | (data_train > (Q3 + 1.5 * IQR))
```

[33]:
```
#Count the number of outliers in each column
print("Number of outliers in each column:")
print(outliers.sum())
```

```
Number of outliers in each column:
count                          3157
diff_srv_rate                  7788
dst_bytes                     23579
dst_host_count                    0
dst_host_diff_srv_rate        10550
dst_host_rerror_rate          22794
dst_host_same_src_port_rate   25051
dst_host_same_srv_rate            0
dst_host_serror_rate              0
dst_host_srv_count                0
dst_host_srv_diff_host_rate   11682
dst_host_srv_rerror_rate      19357
dst_host_srv_serror_rate          0
duration                      10018
flag                              0
hot                            2671
is_guest_login                 1187
is_host_login                     1
land                             25
level                          2995
logged_in                         0
num_access_files                371
num_compromised                1286
num_failed_logins               122
num_file_creations              287
num_outbound_cmds                 0
num_root                        649
```

```
num_shells                    47
outcome                        0
protocol_type                  0
rerror_rate                16190
root_shell                   169
same_srv_rate                  0
serror_rate                    0
service                        0
src_bytes                  13840
srv_count                  12054
srv_diff_host_rate         28399
srv_rerror_rate            16206
srv_serror_rate                0
su_attempted                  80
urgent                         9
wrong_fragment              1090
dtype: int64
```

[34]:
```
# stats contains statistical summary (e.g. output of data_train.describe()),
# extract the mean and standard deviation ('std') rows and store them in␣
 ↪variables 'mean' and 'std_dev' respectively.
mean = stats.loc['mean']
# Calculate the median of each column in the 'data_train' DataFrame and store␣
 ↪it in a variable 'median'.
median = data_train.median()
# Extract the standard deviation from the 'stats' DataFrame.
std_dev = stats.loc['std']
```

[35]:
```
# Print the statistics of each variable
print("Descriptive statistics for each variable:")
print(mean)
print(median)
print(std_dev)
```

```
Descriptive statistics for each variable:
duration                     287.146929
src_bytes                  45567.100824
dst_bytes                  19779.271433
land                           0.000198
wrong_fragment                 0.022688
urgent                         0.000111
hot                            0.204411
num_failed_logins              0.001222
logged_in                      0.395739
num_compromised                0.279253
root_shell                     0.001342
su_attempted                   0.001103
num_root                       0.302194
```

```
num_file_creations               0.012669
num_shells                       0.000413
num_access_files                 0.004096
num_outbound_cmds                0.000000
is_host_login                    0.000008
is_guest_login                   0.009423
count                           84.108207
srv_count                       27.738093
serror_rate                      0.284487
srv_serror_rate                  0.282488
rerror_rate                      0.119959
srv_rerror_rate                  0.121184
same_srv_rate                    0.660925
diff_srv_rate                    0.063053
srv_diff_host_rate               0.097322
dst_host_count                 182.149200
dst_host_srv_count             115.653725
dst_host_same_srv_rate           0.521244
dst_host_diff_srv_rate           0.082952
dst_host_same_src_port_rate      0.148379
dst_host_srv_diff_host_rate      0.032543
dst_host_serror_rate             0.284455
dst_host_srv_serror_rate         0.278487
dst_host_rerror_rate             0.118832
dst_host_srv_rerror_rate         0.120241
level                           19.504056
Name: mean, dtype: float64
duration                         0.00
src_bytes                       44.00
dst_bytes                        0.00
land                             0.00
wrong_fragment                   0.00
urgent                           0.00
hot                              0.00
num_failed_logins                0.00
logged_in                        0.00
num_compromised                  0.00
root_shell                       0.00
su_attempted                     0.00
num_root                         0.00
num_file_creations               0.00
num_shells                       0.00
num_access_files                 0.00
num_outbound_cmds                0.00
is_host_login                    0.00
is_guest_login                   0.00
count                           14.00
srv_count                        8.00
```

```
serror_rate                    0.00
srv_serror_rate                0.00
rerror_rate                    0.00
srv_rerror_rate                0.00
same_srv_rate                  1.00
diff_srv_rate                  0.00
srv_diff_host_rate             0.00
dst_host_count               255.00
dst_host_srv_count            63.00
dst_host_same_srv_rate         0.51
dst_host_diff_srv_rate         0.02
dst_host_same_src_port_rate    0.00
dst_host_srv_diff_host_rate    0.00
dst_host_serror_rate           0.00
dst_host_srv_serror_rate       0.00
dst_host_rerror_rate           0.00
dst_host_srv_rerror_rate       0.00
level                         20.00
dtype: float64
duration                   2.604526e+03
src_bytes                  5.870354e+06
dst_bytes                  4.021285e+06
land                       1.408613e-02
wrong_fragment             2.535310e-01
urgent                     1.436608e-02
hot                        2.149977e+00
num_failed_logins          4.523932e-02
logged_in                  4.890107e-01
num_compromised            2.394214e+01
root_shell                 3.660299e-02
su_attempted               4.515456e-02
num_root                   2.439971e+01
num_file_creations         4.839370e-01
num_shells                 2.218122e-02
num_access_files           9.936995e-02
num_outbound_cmds          0.000000e+00
is_host_login              2.817494e-03
is_guest_login             9.661271e-02
count                      1.145088e+02
srv_count                  7.263609e+01
serror_rate                4.464567e-01
srv_serror_rate            4.470236e-01
rerror_rate                3.204366e-01
srv_rerror_rate            3.236483e-01
same_srv_rate              4.396236e-01
diff_srv_rate              1.803150e-01
srv_diff_host_rate         2.598314e-01
dst_host_count             9.920657e+01
```

```
dst_host_srv_count              1.107029e+02
dst_host_same_srv_rate          4.489501e-01
dst_host_diff_srv_rate          1.889225e-01
dst_host_same_src_port_rate     3.089984e-01
dst_host_srv_diff_host_rate     1.125642e-01
dst_host_serror_rate            4.447851e-01
dst_host_srv_serror_rate        4.456702e-01
dst_host_rerror_rate            3.065586e-01
dst_host_srv_rerror_rate        3.194605e-01
level                           2.291512e+00
Name: std, dtype: float64
```

[36]:
```
# Call the 'describe' method on the DataFrame 'data_train' to get the␣
 ↪statistical summary (e.g. mean, standard deviation).
# Use 'style.background_gradient' to add a background color gradient (using the␣
 ↪color map 'Blues') to the summary table.
# Set the font family of the table to 'Segoe UI'.
data_train.describe().style.background_gradient(cmap='Blues').
 ↪set_properties(**{'font-family':'Segoe UI'})
```

[36]:
```
<pandas.io.formats.style.Styler at 0x1c8542ffa30>
```

1.0.10 Exploratory Analysis (Codes):

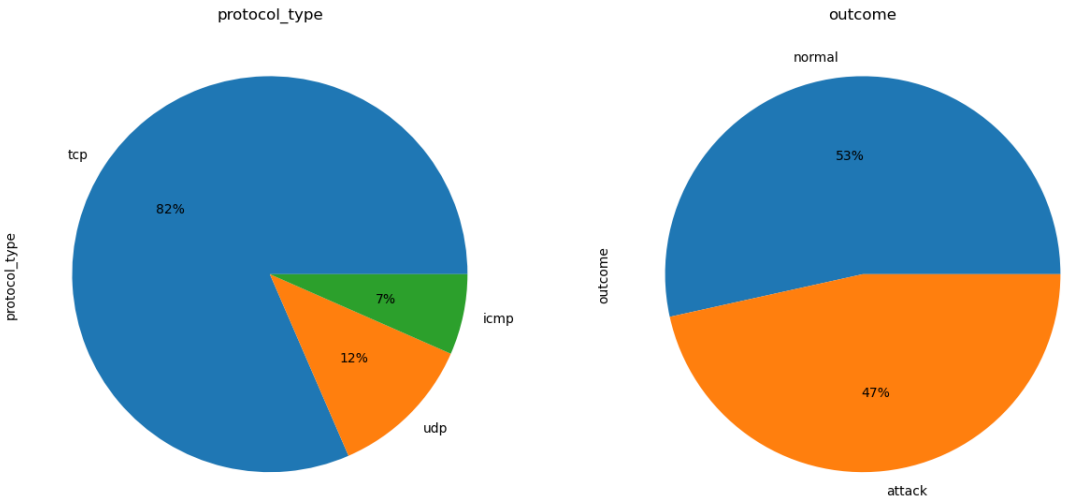
In the following cells, add all the codes used for the exploratory data analysis. The codes should be error-free and return output graphs without any issues. For this analysis, choose appropriate variables from your dataset while considering the business problem you introduced earlier. Generate at least three graphs. [15 Marks]

[37]:
```
# In the DataFrame 'data_train', find all the rows where the 'outcome' column␣
 ↪has the value "normal",
# and make sure they are labeled as 'normal'.
data_train.loc[data_train['outcome'] == "normal", "outcome"] = 'normal'
# In the DataFrame 'data_train', find all the rows where the 'outcome' column␣
 ↪has a value different from "normal",
# and label them as 'attack'. This groups all the non-normal outcomes under a␣
 ↪single label 'attack'.
data_train.loc[data_train['outcome'] != 'normal', "outcome"] = 'attack'
```

[38]:
```
# Define a function called 'pie_plot' that takes four inputs: a DataFrame 'df',␣
 ↪a list of column names 'cols_list',
# and the number of rows 'rows' and columns 'cols' for arranging the plots.
def pie_plot(df, cols_list, rows, cols):
    # Create a figure and a set of subplots with the given number of rows and␣
 ↪columns.
    fig, axes = plt.subplots(rows, cols)
    # Loop through each axis and column name.
```

```
    # 'axes.ravel()' returns a flattened array of axes, making it easier to␣
 ↪loop through them.
    for ax, col in zip(axes.ravel(), cols_list):
        # Count the unique values in the column 'col', and create a pie plot of␣
 ↪these counts.
        # The 'autopct' parameter adds percentage labels to the pie pieces.
        # The plot will have a specified figure size and font size for␣
 ↪readability.
        data_train[col].value_counts().plot(ax=ax, kind='pie', figsize=(15,␣
 ↪15), fontsize=10, autopct='%1.0f%%')
        # Set the title of the subplot to the name of the column being plotted.
        ax.set_title(str(col), fontsize=12)
    # Display the figure with the pie plots.
    plt.show()
```

[17]:
```
# Call the 'pie_plot' function with four inputs: 'data_train',␣
 ↪['protocol_type', 'outcome'], 1, and 2.
# 'data_train' is the DataFrame containing the data to be plotted.
# ['protocol_type', 'outcome'] is a list of column names whose data will be␣
 ↪used for the pie plots.
# 1 and 2 indicate that there will be 1 row and 2 columns of plots (so, two pie␣
 ↪plots side by side).
pie_plot(data_train, ['protocol_type', 'outcome'], 1, 2)
```



#PREPROCESSING

[18]:
```
# Define a function called 'Scaling' that takes two inputs: a DataFrame␣
 ↪'df_num' and a list of column names 'cols'.
def Scaling(df_num, cols):
```

```
    # Create a StandardScaler object. StandardScaler standardizes features by␣
 ↪removing the mean
    # and scaling to unit variance (also known as z-score normalization).
    std_scaler = StandardScaler()
    # Fit the scaler to the data in 'df_num' and transform it.
    # The transformed data will have a mean value of 0 and a standard deviation␣
 ↪of 1.
    std_scaler_temp = std_scaler.fit_transform(df_num)
    # Convert the transformed data back into a DataFrame.
    # It will have the same column names as the input data 'df_num'.
    std_df = pd.DataFrame(std_scaler_temp, columns=cols)
    # Return the standardized DataFrame.
    return std_df
```

[19]:
```
# List of columns in the dataset that are categorical in nature.
cat_cols = ['is_host_login', 'protocol_type', 'service', 'flag', 'land',␣
 ↪'logged_in', 'is_guest_login', 'level', 'outcome']
# Define a function named 'preprocess' that takes a DataFrame as an input and␣
 ↪returns a preprocessed DataFrame.
def preprocess(dataframe):
    # Drop the categorical columns from the DataFrame and keep the numerical␣
 ↪columns.
    # 'df_num' now contains only the numerical columns.
    df_num = dataframe.drop(cat_cols, axis=1)
    # Store the names of the numerical columns in a variable 'num_cols'.
    num_cols = df_num.columns
    # Scale the numerical features using a custom function named 'Scaling'.
    # This function is assumed to be defined somewhere else in the code.
    # Store the scaled data in 'scaled_df'.
    scaled_df = Scaling(df_num, num_cols)
    # Remove the original (unscaled) numerical columns from the input DataFrame.
    dataframe.drop(labels=num_cols, axis="columns", inplace=True)
    # Add the scaled numerical columns back to the input DataFrame.
    dataframe[num_cols] = scaled_df[num_cols]
    # Change the 'outcome' column to be binary.
    # Set the 'outcome' column to 0 if it's "normal", and 1 otherwise.
    dataframe.loc[dataframe['outcome'] == "normal", "outcome"] = 0
    dataframe.loc[dataframe['outcome'] != 0, "outcome"] = 1
    # Convert categorical columns ('protocol_type', 'service', 'flag') into␣
 ↪dummy/indicator variables.
    # This is necessary for machine learning algorithms to work with␣
 ↪categorical data.
    dataframe = pd.get_dummies(dataframe, columns=['protocol_type', 'service',␣
 ↪'flag'])
    # Return the preprocessed DataFrame.
    return dataframe
```

[20]:
```
# Call the 'preprocess' function with the 'data_train' DataFrame as an input.
# The 'preprocess' function is expected to perform several data preprocessing␣
 ↪steps such as scaling numerical features,
# binarizing a target column, and converting categorical variables into dummy/
 ↪indicator variables.
# Store the resulting preprocessed DataFrame in a variable named 'scaled_train'.
scaled_train = preprocess(data_train)
```

#Split the Dataset

[21]:
```
# Extract feature data (x) from 'scaled_train' DataFrame by dropping 'outcome'␣
 ↪and 'level' columns.
# The '.values' extracts the data in the form of a NumPy array.
x = scaled_train.drop(['outcome', 'level'], axis=1).values
# Extract the target labels for classification (y) from the 'outcome' column of␣
 ↪the 'scaled_train' DataFrame.
y = scaled_train['outcome'].values
# Extract the target labels for regression (y_reg) from the 'level' column of␣
 ↪the 'scaled_train' DataFrame.
y_reg = scaled_train['level'].values
# Convert the datatype of the target labels for classification (y) to integers.
# This is important as classification algorithms expect integer labels.
y = y.astype('int')
# Split the data into training and a temporary set using 20% of the data for␣
 ↪the temporary set.
# 'shuffle=True' means the data will be shuffled before splitting.
# 'random_state' ensures that the splits generate are reproducible.
x_train, x_, y_train, y_ = train_test_split(x, y, shuffle=True, test_size=0.2,␣
 ↪random_state=42)
# Further split the temporary set into validation and test sets (both␣
 ↪containing 50% of the temporary set).
X_val, X_test, y_val, y_test = train_test_split(x_, y_, shuffle=True,␣
 ↪test_size=0.5, random_state=42)
# Split the data into training and testing sets for regression task using 20%␣
 ↪of the data for the test set.
x_train_reg, x_test_reg, y_train_reg, y_test_reg = train_test_split(x, y_reg,␣
 ↪shuffle=True, test_size=0.2, random_state=42)
# Print the shape of the training set for classification to see how many␣
 ↪samples and features it contains.
print(x_train.shape)
# Print the shape of the validation set to see how many samples and features it␣
 ↪contains.
print(X_val.shape)
# Print the shape of the test set for classification to see how many samples␣
 ↪and features it contains.
print(X_test.shape)
```

```
(100777, 122)
(12597, 122)
(12598, 122)
```

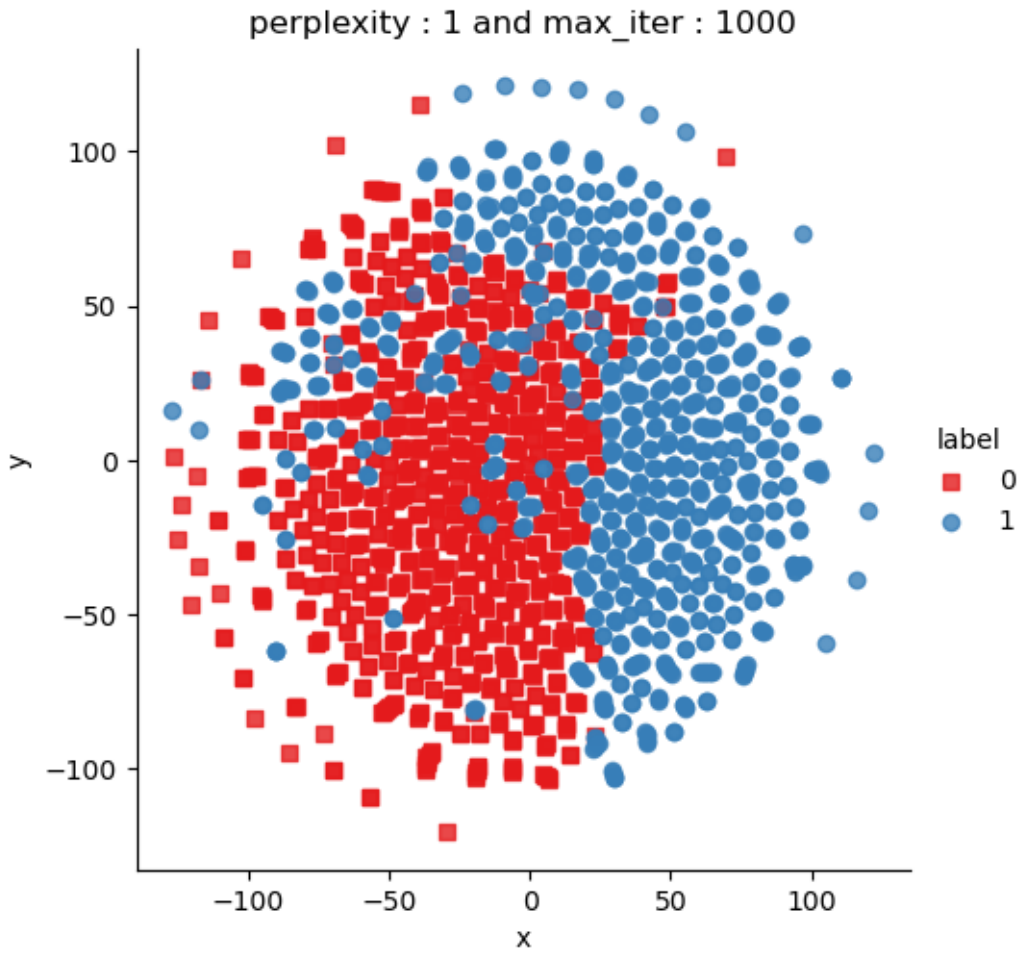
[22]:
```python
def perform_tsne(X_data, y_data, perplexities, n_iter=1000,␣
↪img_name_prefix='t-sne'):
    # This function performs t-SNE (t-Distributed Stochastic Neighbor␣
↪Embedding) on the data.
    # It's used for reducing high-dimensional data and for visualization in 2D.
    # X_data is the high-dimensional data, y_data contains the labels, and␣
↪perplexities is a list of perplexity values
    # to be used in t-SNE. n_iter is the maximum number of iterations for␣
↪optimization (default is 1000).
    # img_name_prefix is used as the prefix for the image filenames where the␣
↪plots will be saved.
    # Loop through each perplexity value provided in the 'perplexities' list.
    for index, perplexity in enumerate(perplexities):
        # Perform t-SNE on the data with the given perplexity.
        # The verbose parameter set to 2 will show more messages during the␣
↪optimization process.
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        # Prepare the t-SNE results to be plotted.
        # Create a DataFrame (like a table) with columns 'x' and 'y' for the 2D␣
↪t-SNE components,
        # and 'label' for the corresponding labels in y_data.
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x': X_reduced[:, 0], 'y': X_reduced[:, 1], 'label':␣
↪y_data})
        # Create a scatter plot of the t-SNE results.
        # Each point on the plot represents a sample, and they are colored␣
↪based on their label.
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False,␣
↪palette="Set1", markers=['s', 'o'])
        # Set the title of the plot showing the used perplexity and maximum␣
↪number of iterations.
        plt.title("perplexity : {} and max_iter : {}".format(perplexity,␣
↪n_iter))
        # Set the filename for the image where the plot will be saved.
        # It uses the prefix given in 'img_name_prefix' and adds information␣
↪about the perplexity and iterations.
        img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity,␣
↪n_iter)
        # Save the plot as an image file in the current working directory.
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        # Display the plot in the output.
```

```
        plt.show()
        # Print "Done" to indicate that the plot for this perplexity value has␣
 ↪been created and saved.
        print('Done')
```

[23]:
```
# Call the 'perform_tsne' function to perform t-SNE (t-Distributed Stochastic␣
 ↪Neighbor Embedding) on the data.
# t-SNE is a technique for reducing the dimensions of data, typically used for␣
 ↪visualization.
# 'X_data' is set to the first 2000 samples of 'x', which represents the␣
 ↪feature data.
# 'y_data' is set to the first 2000 samples of 'y', which represents the␣
 ↪corresponding labels or targets.
# 'perplexities' is set to [1], which is a parameter that affects the balance␣
 ↪between preserving the
# local and global structure of the data. In this case, it is set to a list␣
 ↪with a single value of 1.
perform_tsne(X_data=x[:2000], y_data=y[:2000], perplexities=[1])
```

```
[t-SNE] Computing 4 nearest neighbors…
[t-SNE] Indexed 2000 samples in 0.002s…
[t-SNE] Computed neighbors for 2000 samples in 0.175s…
[t-SNE] Computed conditional probabilities for sample 1000 / 2000
[t-SNE] Computed conditional probabilities for sample 2000 / 2000
[t-SNE] Mean sigma: 0.000114
[t-SNE] Computed conditional probabilities in 0.003s
[t-SNE] Iteration 50: error = 110.8958817, gradient norm = 0.0991733 (50
iterations in 0.280s)
[t-SNE] Iteration 100: error = 94.8509369, gradient norm = 0.0801058 (50
iterations in 0.181s)
[t-SNE] Iteration 150: error = 88.4697418, gradient norm = 0.0574595 (50
iterations in 0.177s)
[t-SNE] Iteration 200: error = 84.4951553, gradient norm = 0.0380292 (50
iterations in 0.177s)
[t-SNE] Iteration 250: error = 81.6376343, gradient norm = 0.0425930 (50
iterations in 0.181s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 81.637634
[t-SNE] Iteration 300: error = 3.2317238, gradient norm = 0.0324813 (50
iterations in 0.259s)
[t-SNE] Iteration 350: error = 2.3204095, gradient norm = 0.0351024 (50
iterations in 0.230s)
[t-SNE] Iteration 400: error = 1.9437399, gradient norm = 0.0223982 (50
iterations in 0.178s)
[t-SNE] Iteration 450: error = 1.6849772, gradient norm = 0.0223110 (50
iterations in 0.189s)
[t-SNE] Iteration 500: error = 1.5150521, gradient norm = 0.0173773 (50
iterations in 0.186s)
```

```
[t-SNE] Iteration 550: error = 1.3860444, gradient norm = 0.0153243 (50
iterations in 0.194s)
[t-SNE] Iteration 600: error = 1.2877619, gradient norm = 0.0143866 (50
iterations in 0.184s)
[t-SNE] Iteration 650: error = 1.2065322, gradient norm = 0.0133581 (50
iterations in 0.187s)
[t-SNE] Iteration 700: error = 1.1389191, gradient norm = 0.0128622 (50
iterations in 0.181s)
[t-SNE] Iteration 750: error = 1.0813384, gradient norm = 0.0121058 (50
iterations in 0.185s)
[t-SNE] Iteration 800: error = 1.0322831, gradient norm = 0.0113614 (50
iterations in 0.202s)
[t-SNE] Iteration 850: error = 0.9894409, gradient norm = 0.0106460 (50
iterations in 0.190s)
[t-SNE] Iteration 900: error = 0.9527267, gradient norm = 0.0102558 (50
iterations in 0.183s)
[t-SNE] Iteration 950: error = 0.9187794, gradient norm = 0.0099013 (50
iterations in 0.204s)
[t-SNE] Iteration 1000: error = 0.8895234, gradient norm = 0.0093649 (50
iterations in 0.261s)
[t-SNE] KL divergence after 1000 iterations: 0.889523
Creating plot for this t-sne visualization..
saving this plot as image in present working directory…
```




```
Done
```

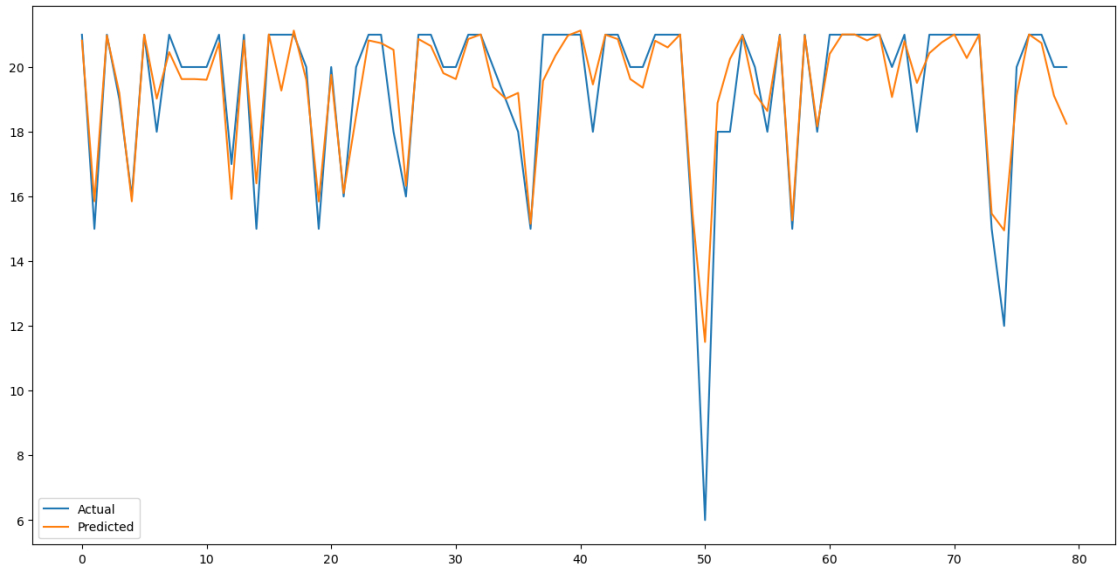
#MODELLING

[24]:
```
# Set the variable 'name' to the text "XGBOOST", which represents the name of␣
 ↪the algorithm being used.
name = "XGBOOST"
# Create an XGBoost Regressor model.
# This is a popular machine learning model used for regression tasks␣
 ↪(predicting a continuous value).
# 'objective' is set to 'reg:linear' indicating that we are solving a linear␣
 ↪regression problem.
# 'n_estimators' is set to 20, which means the model will have 20 decision␣
 ↪trees in the ensemble.
# The model is then trained using the training data 'x_train_reg' and target␣
 ↪values 'y_train_reg'.
```

```
xg_r = xgb.XGBRegressor(objective='reg:linear', n_estimators=20).
 ↪fit(x_train_reg, y_train_reg)
# Calculate the training error using the mean squared error.
# This represents how well the model fits the training data.
# A lower value is better and represents a smaller error.
# The 'squared' parameter set to False means it calculates the root mean␣
 ↪squared error (RMSE).
train_error = metrics.mean_squared_error(y_train_reg, xg_r.
 ↪predict(x_train_reg), squared=False)
# Calculate the test error, which represents how well the model does on unseen␣
 ↪data.
# Like the training error, a lower test error is better.
test_error = metrics.mean_squared_error(y_test_reg, xg_r.predict(x_test_reg),␣
 ↪squared=False)
# Print the training and test errors to the screen.
print("Training Error " + str(name) + " {}  Test error ".format(train_error) +␣
 ↪str(name) + " {}".format(test_error))
# Make predictions on the test data.
y_pred = xg_r.predict(x_test_reg)
# Create a DataFrame to organize the actual target values ('Y_test') and␣
 ↪predicted values ('Y_pred').
df = pd.DataFrame({"Y_test": y_test_reg, "Y_pred": y_pred})
# Create a plot to visually compare the actual and predicted values.
# Set the size of the plot for better visibility.
plt.figure(figsize=(16, 8))
# Plot the data.
# The '[:80]' means that we are only plotting the first 80 data points.
plt.plot(df[:80])
# Add a legend to the plot to indicate which line is actual and which is␣
 ↪predicted.
plt.legend(['Actual', 'Predicted'])
```

```
[19:03:45] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-
windows\src\objective\regression_obj.cu:213: reg:linear is now deprecated in
favor of reg:squarederror.
Training Error XGBOOST 0.9549338655457729  Test error XGBOOST 1.0322196273094075
```

[24]: <matplotlib.legend.Legend at 0x1b72d73e590>



#NEURAL NETWORKS

##Visulazition function

[25]:
```
# Define a function named 'plot_loss' that takes in two arguments: 'histories' and 'title'.
def plot_loss(histories, title):
    # Create an empty list named 'his_df' to store the validation loss values.
    his_df = []
    # Append the last validation loss value from the first element of 'histories' to 'his_df'.
    his_df.append(histories[0].history['val_loss'][-1])
    # Loop through each 'history' in 'histories'.
    for history in histories:
        # If the last validation loss value of the current 'history' is lower than
        # the last value in 'his_df', append it to 'his_df'.
        if history.history['val_loss'][-1] < his_df[-1]:
            his_df.append(history.history['val_loss'][-1])
        # If the current validation loss is not lower, do nothing and continue the loop.
        else:
            continue
    # Plot the values in 'his_df' as a line graph, and label this line as 'val_loss'.
    plt.plot(his_df, label='val_loss')
    # Label the x-axis as 'Epoch'.
    plt.xlabel('Epoch')
```

```
    # Label the y-axis as 'Loss'.
    plt.ylabel('Loss')
    # Set the title of the plot to the provided 'title'.
    plt.title(title)
    # Display the legend to identify what the lines represent.
    plt.legend()
    # Display grid lines on the plot for better readability.
    plt.grid(True)
    # Show the plot.
    plt.show()
```

#Approach Implementation (TensorFlow Approximation)

[26]:
```
# Initialize an empty list to store the history of model training
history_list = []
# Define a function to train a model over a specified number of episodes
def train(episodes):
    # Build a Linear Support Vector Classifier and fit it to the training data
    lr = svm.LinearSVC().fit(x_train, y_train)
    # Calculate the hinge loss of the linear classifier on the test data
    lin_loss = metrics.hinge_loss(y_test, lr.predict(X_test))
    # Initialize a variable to store the current model
    current_model = None
    # Set the maximum number of episodes
    max_episodes = 32
    # Loop over the number of episodes
    for i in range(episodes):
        # Print the current episode number
        print("Entering Episode {}".format(i + 1))
        # Define the input layer of the neural network, which has the same␣
 ↪shape as the training data
        input_layer = tf.keras.layers.Input(shape=(x_train.shape[1:]))
        # Define a dense layer with a single unit and a ReLU activation function
        # The weights are initialized with a normal distribution and␣
 ↪constrained to be non-negative
        current_base_layer = tf.keras.layers.Dense(units=1, activation='relu',
                             kernel_initializer=initializers.RandomNormal(mean=0.
 ↪0, stddev=1.0, seed=42),
                             kernel_constraint=tf.keras.constraints.NonNeg())
        # Connect the input layer to the base layer
        current_base_layer = current_base_layer(input_layer)
        # Define another dense layer with a single unit and a ReLU activation␣
 ↪function
        unit = tf.keras.layers.Dense(units=1, activation='relu',
                             kernel_initializer=initializers.RandomNormal(mean=0.
 ↪0, stddev=1.0, seed=42),
```

```
                        kernel_constraint=tf.keras.constraints.
↪NonNeg())(input_layer)
        # Connect the base layer to the unit layer
        x = current_base_layer
        for j in range(i):
            x = tf.keras.layers.Concatenate()([x, unit])
        # Update the base layer
        current_base_layer = x
        # Define the output layer with a single unit and a sigmoid activation␣
↪function
        output = tf.keras.layers.Dense(1, activation='sigmoid',␣
↪kernel_regularizer=regularizers.L1())(x)
        # Define the model
        current_model = tf.keras.models.Model(inputs=input_layer,␣
↪outputs=output)
        # Compile the model with an Adam optimizer and hinge loss
        current_model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.
↪keras.losses.Hinge())
        # Plot the model architecture and save it as a PNG file
        tf.keras.utils.plot_model(current_model, show_shapes=True, to_file =␣
↪str(i) + '.png', show_layer_names=True)
        # Fit the model to the training data and validate it on the validation␣
↪data
        current_history = current_model.fit(x_train, y_train,␣
↪validation_data=(X_val, y_val), epochs=i+1, verbose=0)
        # Append the history to the history list
        history_list.append(current_history)
        # Evaluate the model on the test data
        nn_loss = current_model.evaluate(X_test, y_test)
        # Print the losses of the linear classifier and the neural network
        print("Linear Classifier Loss: {} Neural Network Loss: {}".
↪format(lin_loss, nn_loss))
        # If the current episode number is less than the maximum number of␣
↪episodes
        if i < max_episodes:
            # If the loss of the linear classifier is less than the loss of the␣
↪neural network
            if lin_loss < nn_loss:
                # Continue to the next iteration of the loop without executing␣
↪the rest of the code in the loop
                pass
            # If the loss of the linear classifier is greater than or equal to␣
↪the loss of the neural network
            elif lin_loss >= nn_loss:
                # Break the loop and stop training
                break
```

```
        # If the current episode number is equal to the maximum number of␣
 ↪episodes
        else:
            # Break the loop and stop training
            break
    # Print the optimum weights of the model
    print("Optimum weights are: ", current_model.get_weights()[0])
```

[27]:
```
# Set the number of episodes to 64.
# In this context, an "episode" refers to a complete cycle of training or␣
 ↪simulation.
episodes = 64
# Call the 'train' function and tell it to perform training for 64 episodes.
# The 'train' function is a mini-program designed to train a machine learning␣
 ↪model or run simulations.
# It takes the number of episodes as input, which tells it how many cycles of␣
 ↪training or simulation to perform.
train(episodes)
```

```
Entering Episode 1
394/394 [==============================] - 0s 766us/step - loss: 0.6569
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.6569440364837646
Entering Episode 2
394/394 [==============================] - 0s 719us/step - loss: 0.5994
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5994184613227844
Entering Episode 3
394/394 [==============================] - 0s 779us/step - loss: 0.6112
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.6111689805984497
Entering Episode 4
394/394 [==============================] - 0s 719us/step - loss: 0.6083
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.6083024144172668
Entering Episode 5
394/394 [==============================] - 0s 721us/step - loss: 0.5796
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5795797109603882
Entering Episode 6
394/394 [==============================] - 0s 767us/step - loss: 0.5802
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5801517963409424
Entering Episode 7
394/394 [==============================] - 0s 722us/step - loss: 0.5786
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5785994529724121
```

```
Entering Episode 8
394/394 [==============================] - 0s 692us/step - loss: 0.5778
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5778390169143677
Entering Episode 9
394/394 [==============================] - 0s 716us/step - loss: 0.5766
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5765815377235413
Entering Episode 10
394/394 [==============================] - 0s 736us/step - loss: 0.5761
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5761493444442749
Entering Episode 11
394/394 [==============================] - 0s 747us/step - loss: 0.5752
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5751587152481079
Entering Episode 12
394/394 [==============================] - 0s 846us/step - loss: 0.5745
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5745184421539307
Entering Episode 13
394/394 [==============================] - 0s 822us/step - loss: 0.5742
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5741586685180664
Entering Episode 14
394/394 [==============================] - 0s 737us/step - loss: 0.5737
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5736727714538574
Entering Episode 15
394/394 [==============================] - 0s 739us/step - loss: 0.5732
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.573234498500824
Entering Episode 16
394/394 [==============================] - 0s 764us/step - loss: 0.5729
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.572944700717926
Entering Episode 17
394/394 [==============================] - 0s 746us/step - loss: 0.5725
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5724908709526062
Entering Episode 18
394/394 [==============================] - 0s 756us/step - loss: 0.5726
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5725516080856323
Entering Episode 19
394/394 [==============================] - 0s 827us/step - loss: 0.5720
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5720142126083374
```

```
Entering Episode 20
394/394 [==============================] - 0s 756us/step - loss: 0.5716
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5715900659561157
Entering Episode 21
394/394 [==============================] - 0s 765us/step - loss: 0.5717
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.571681559085846
Entering Episode 22
394/394 [==============================] - 0s 815us/step - loss: 0.5713
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5713458061218262
Entering Episode 23
394/394 [==============================] - 0s 761us/step - loss: 0.5710
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5710358023643494
Entering Episode 24
394/394 [==============================] - 0s 800us/step - loss: 0.5711
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5710828900337219
Entering Episode 25
394/394 [==============================] - 0s 754us/step - loss: 0.5706
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5706371068954468
Entering Episode 26
394/394 [==============================] - 0s 733us/step - loss: 0.5705
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5704522132873535
Entering Episode 27
394/394 [==============================] - 0s 771us/step - loss: 0.5702
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5702377557754517
Entering Episode 28
394/394 [==============================] - 0s 733us/step - loss: 0.5703
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5703414678573608
Entering Episode 29
394/394 [==============================] - 0s 738us/step - loss: 0.5700
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5699934959411621
Entering Episode 30
394/394 [==============================] - 0s 726us/step - loss: 0.5698
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5698093175888062
Entering Episode 31
394/394 [==============================] - 0s 762us/step - loss: 0.5697
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5697152018547058
```

```
Entering Episode 32
394/394 [==============================] - 0s 815us/step - loss: 0.5696
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5695530772209167
Entering Episode 33
394/394 [==============================] - 0s 767us/step - loss: 0.5700
Linear Classifier Loss: 0.5612795681854262 Neural Network Loss:
0.5700396299362183
Optimum weights are:  [[ 2.50688314e+00]
 [-0.00000000e+00]
 [-0.00000000e+00]
 [-0.00000000e+00]
 [ 3.68390717e-02]
 [ 1.03262458e-02]
 [ 1.05611514e-02]
 [ 5.46185398e+00]
 [ 1.12999178e-01]
 [ 1.35988211e-02]
 [ 8.80547047e-01]
 [-0.00000000e+00]
 [ 8.39809421e-03]
 [ 4.48155496e-03]
 [-0.00000000e+00]
 [-0.00000000e+00]
 [ 1.16986623e-02]
 [ 1.23131387e-02]
 [ 1.11511922e+00]
 [ 5.83330059e+00]
 [ 1.20381367e+00]
 [ 4.25461245e+00]
 [ 4.32989120e+00]
 [ 2.81271744e+00]
 [ 1.98320413e+00]
 [ 7.16731045e-03]
 [ 2.37523818e+00]
 [ 1.66495994e-03]
 [ 1.10538139e+01]
 [ 5.15522726e-04]
 [ 4.40028962e-03]
 [ 4.29823351e+00]
 [ 3.94064546e+00]
 [ 1.10892886e-02]
 [ 4.15986490e+00]
 [ 6.66937256e+00]
 [ 4.75258446e+00]
 [ 9.87323225e-01]
 [ 1.59162533e+00]
 [ 5.59717751e+00]
```

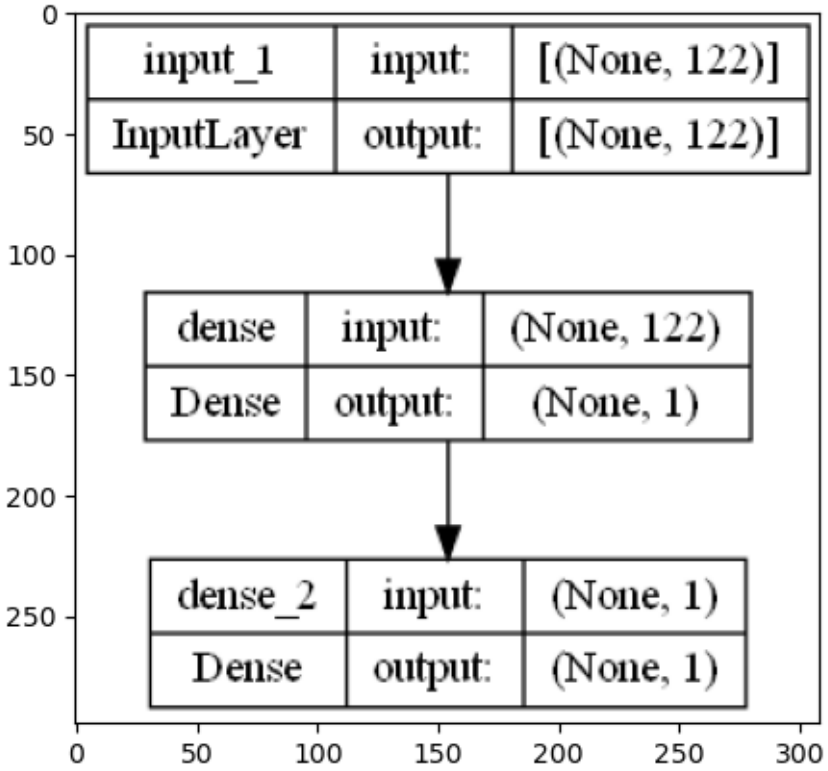
```
[ 2.49336648e+00]
[-0.00000000e+00]
[ 6.66700909e-03]
[ 1.00273800e+01]
[ 5.59728704e-02]
[ 1.50116244e-02]
[ 9.41468334e+00]
[ 1.12872562e+01]
[ 8.09191895e+00]
[ 3.18350196e+00]
[ 4.07763815e+00]
[ 2.89493275e+00]
[ 5.85207796e+00]
[ 4.51867849e-01]
[ 3.64068627e+00]
[-0.00000000e+00]
[ 1.01766605e+01]
[ 1.01516075e+01]
[ 6.37057686e+00]
[ 3.81825113e+00]
[ 1.08310771e+00]
[ 4.08351231e+00]
[ 5.17472649e+00]
[ 1.98513877e+00]
[ 2.96395612e+00]
[ 7.54179945e-03]
[ 6.55949637e-02]
[ 1.07898531e+01]
[ 1.88870358e+00]
[ 3.47692966e+00]
[ 2.00790834e+00]
[ 9.35398960e+00]
[ 8.82834911e+00]
[ 4.19044876e+00]
[ 3.19841623e+00]
[ 5.61312580e+00]
[ 2.12178397e+00]
[ 3.67245245e+00]
[ 5.40853119e+00]
[ 6.00280237e+00]
[ 6.55493355e+00]
[ 3.57290721e+00]
[ 1.09537430e+01]
[ 2.16588926e+00]
[-0.00000000e+00]
[-0.00000000e+00]
[ 1.05178297e+00]
[ 3.35742736e+00]
```

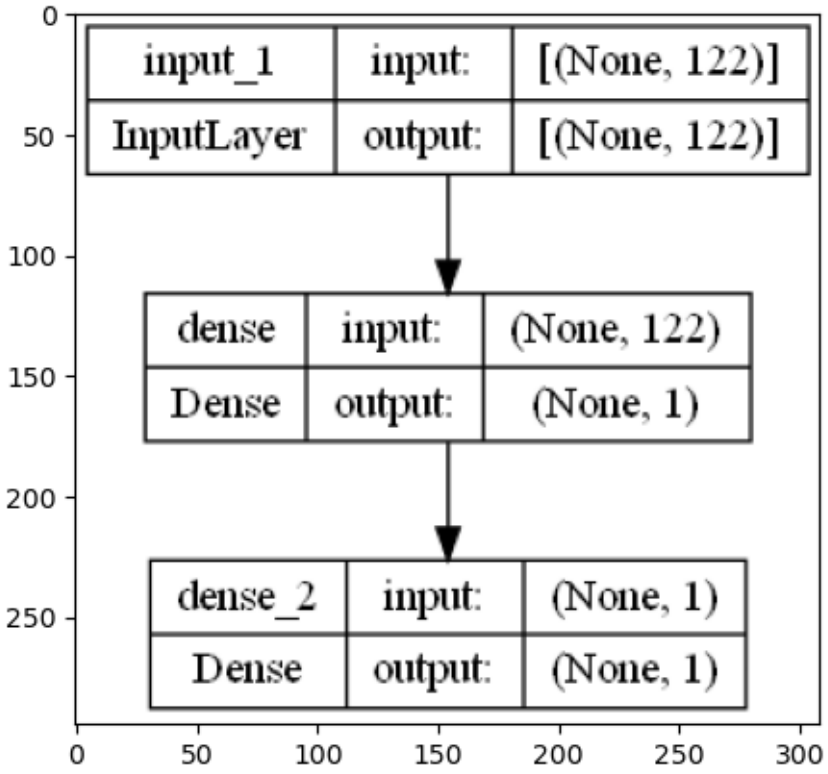
```
[ 2.34642834e-03]
[ 1.20470452e+00]
[ 1.72097015e+01]
[ 5.87671995e-01]
[ 2.13247800e+00]
[ 2.18385482e+00]
[ 1.91602790e+00]
[ 1.41682112e+00]
[ 6.25986481e+00]
[ 5.53735495e+00]
[ 3.77493310e+00]
[ 2.13290167e+00]
[ 4.26638317e+00]
[ 8.02093863e-01]
[ 1.93071115e+00]
[-0.00000000e+00]
[ 5.35117531e+00]
[-0.00000000e+00]
[-0.00000000e+00]
[ 2.65223432e+00]
[ 8.65299797e+00]
[ 8.75572109e+00]
[ 3.51952410e+00]
[ 2.91438103e+00]
[ 4.82037254e-02]
[ 3.29329848e+00]
[ 2.69639492e+00]
[ 4.80974817e+00]
[ 1.09943457e+01]
[-0.00000000e+00]
[ 1.00496163e-06]
[ 3.94173674e-02]
[ 1.08624661e+00]
[ 2.21150851e+00]]
```

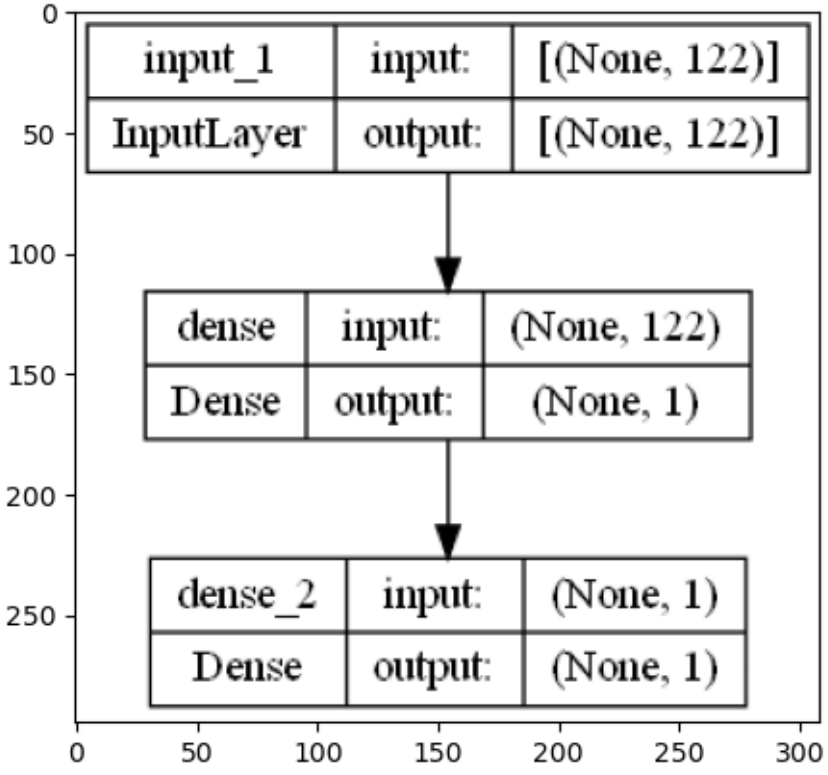
[28]:
```python
# Define a function named 'disp_models' that takes in one argument: 'episodes'.
def disp_models(episodes):
    # Loop starting from 0 up to the number of 'episodes' specified.
    for i in range(episodes):
        # Load an image from the file named '0.png'. Note that '.format(i)' seems to be intended to insert
        # the current value of 'i' into the filename, but it's not actually used.
        # So, it's always loading the same image called '0.png'.
        img = plt.imread('0.png'.format(i)) # your image loading here
        # Display the image that has been loaded.
        plt.imshow(img)
```

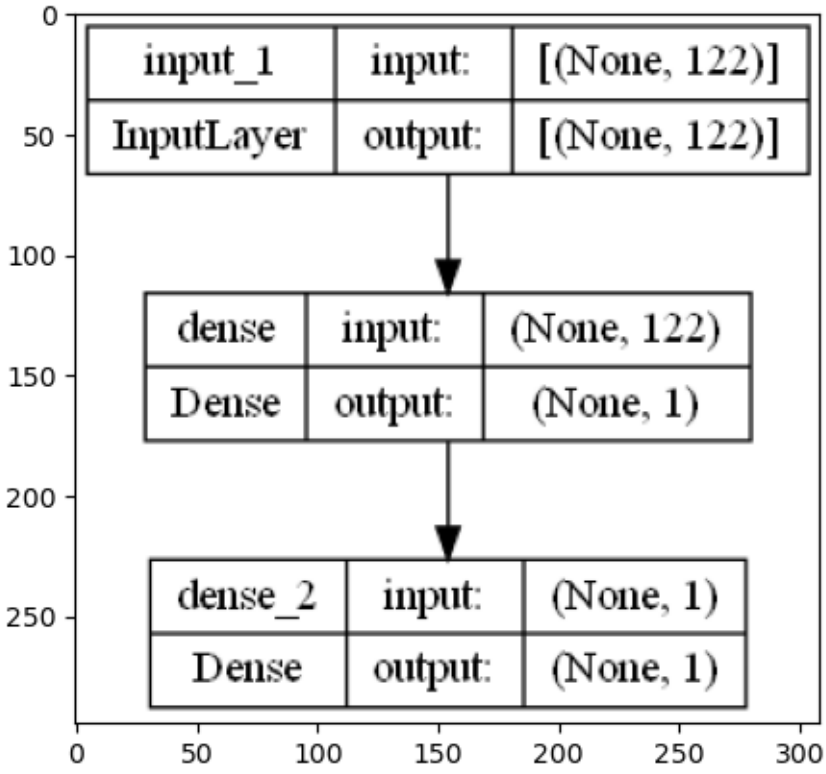
```
        # Show the image on the screen.
        plt.show()
```

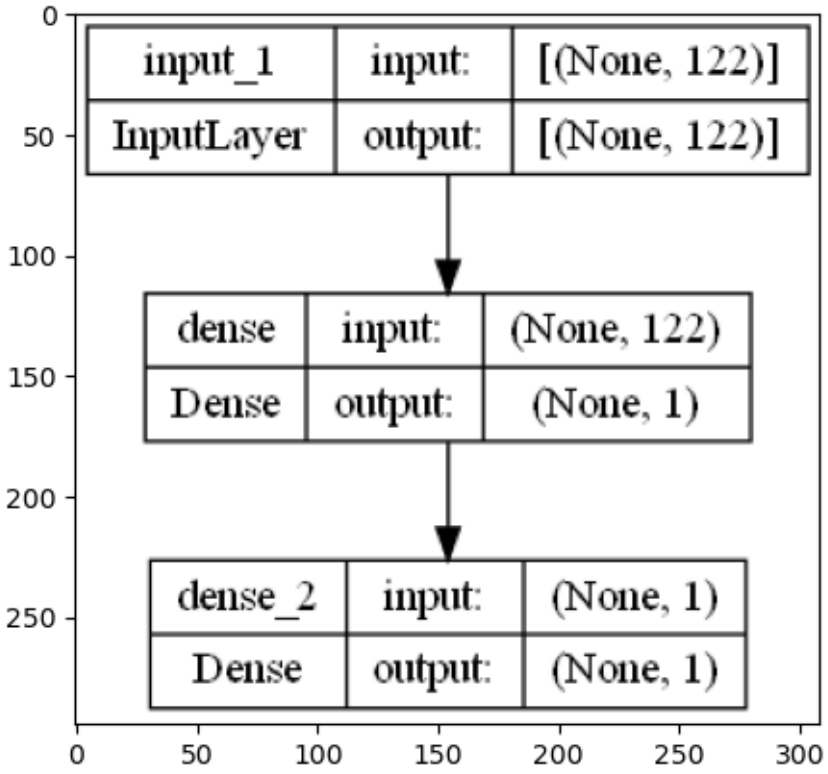
[29]:
```
# Call the 'disp_models' function to display images for each entry in␣
↪'history_list'.
# The number of images to be displayed is equal to the length of 'history_list'.
disp_models(len(history_list))
```

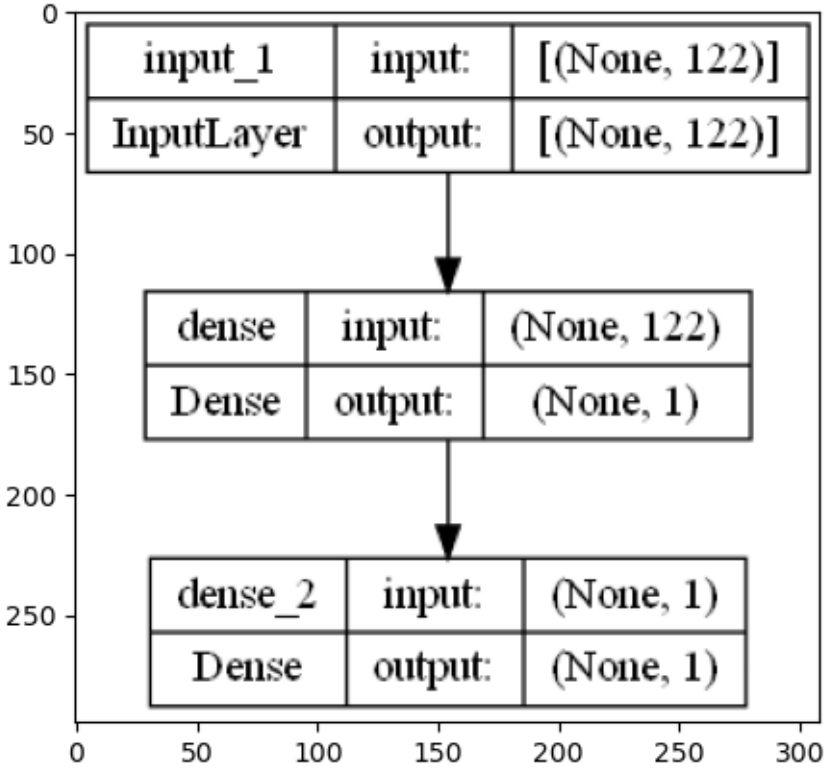


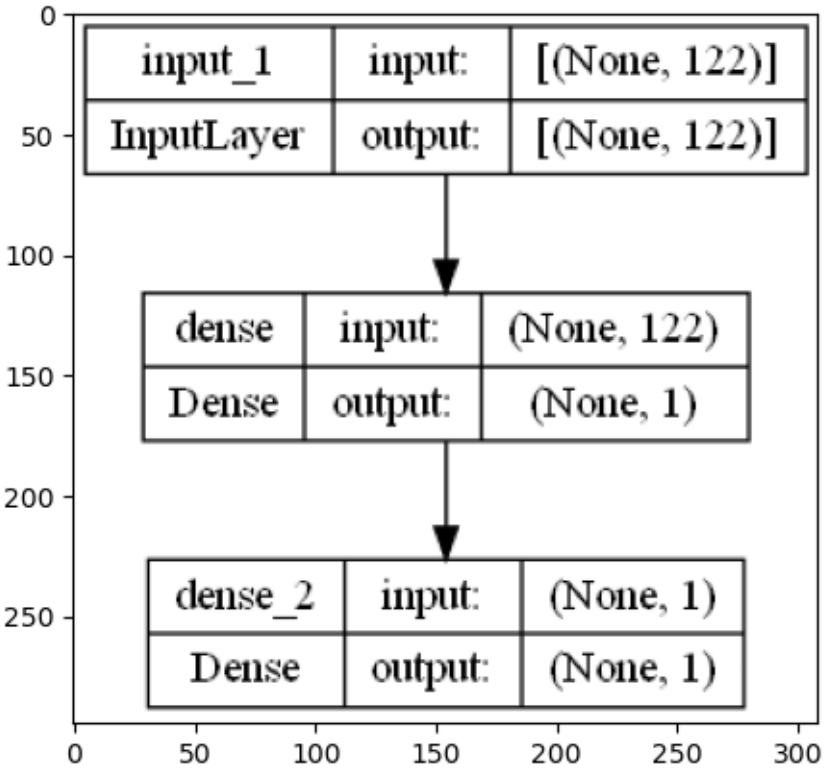

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

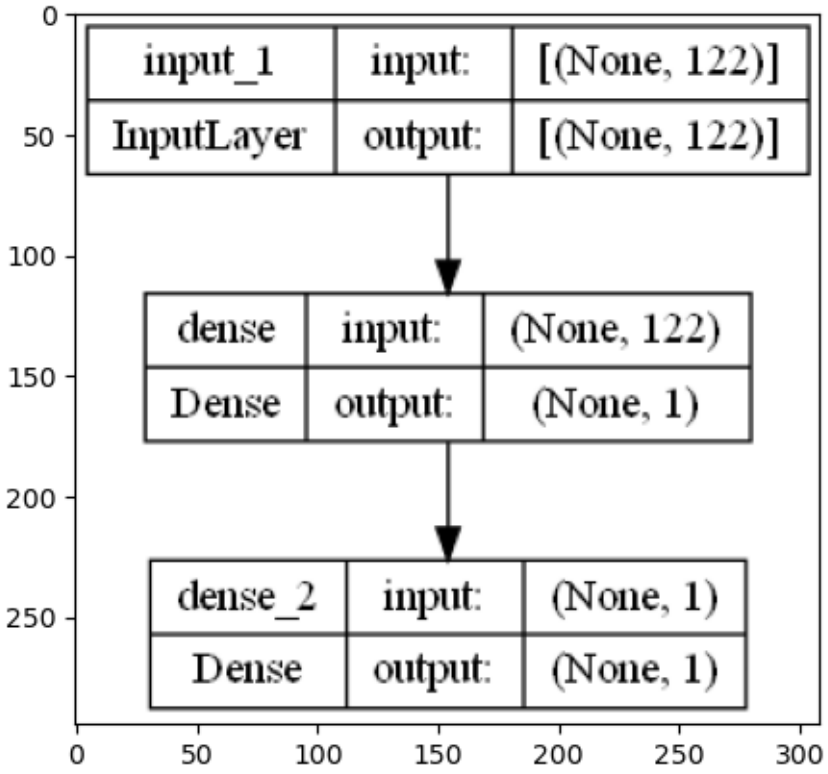

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

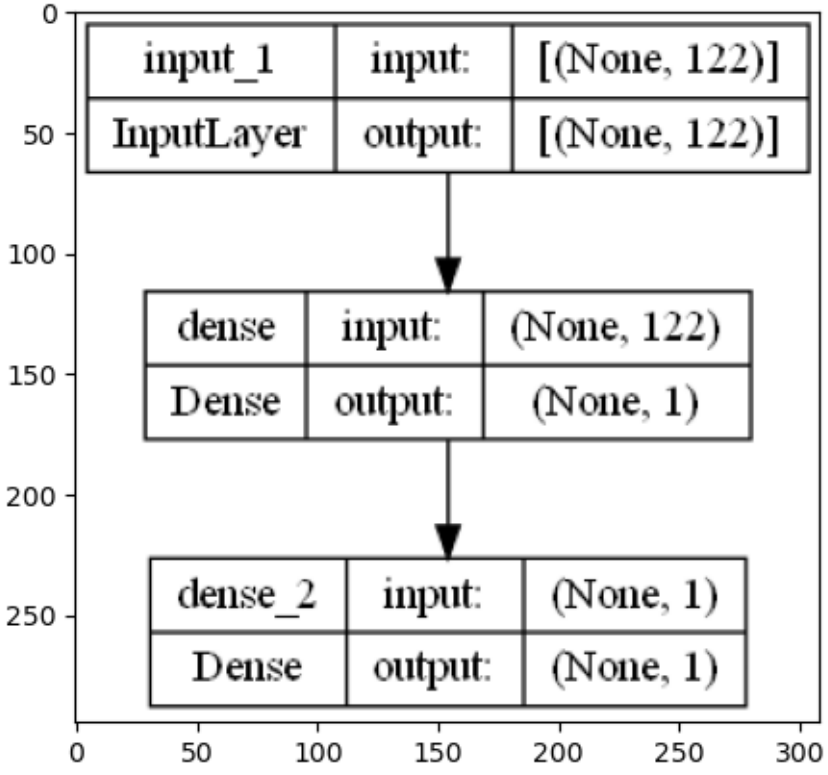

input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
300

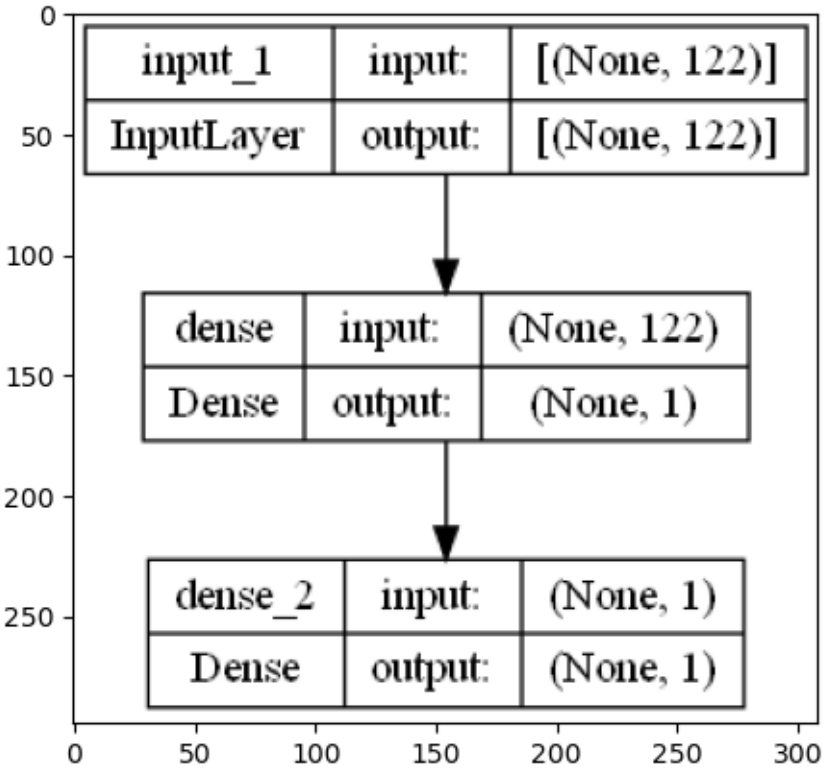

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

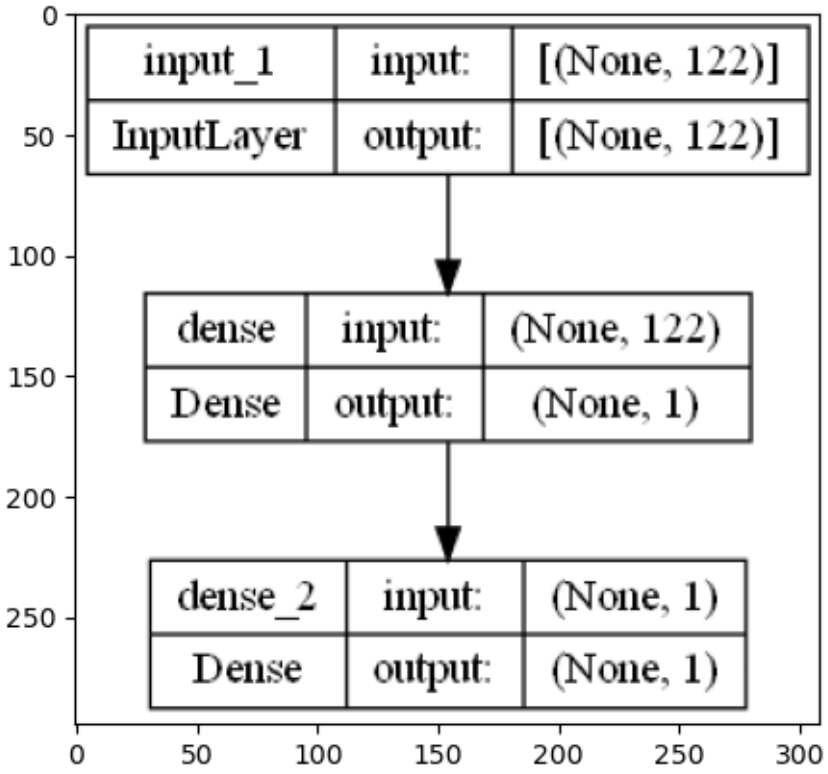

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

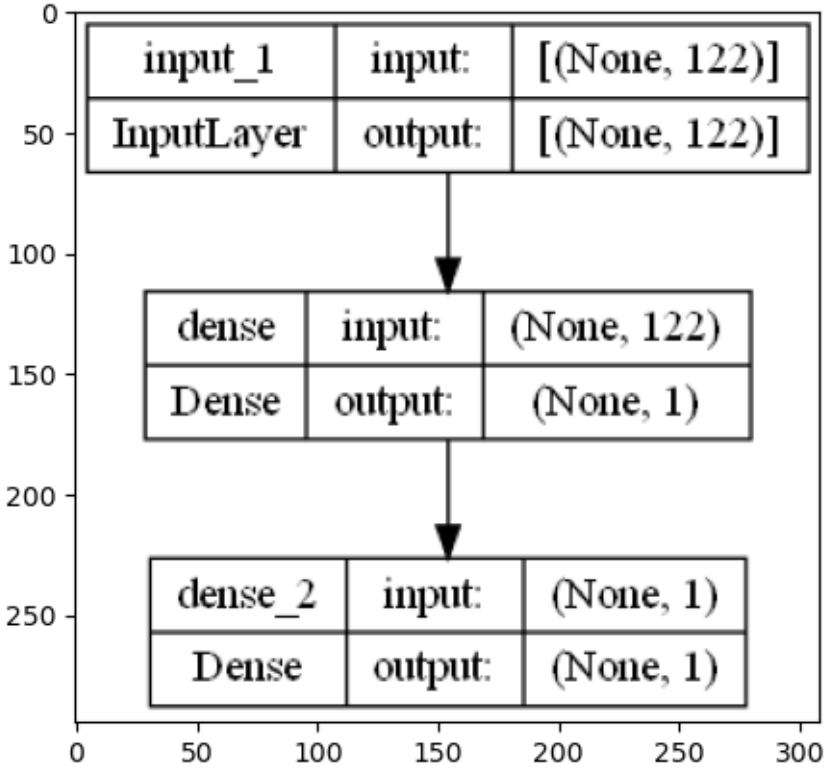

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

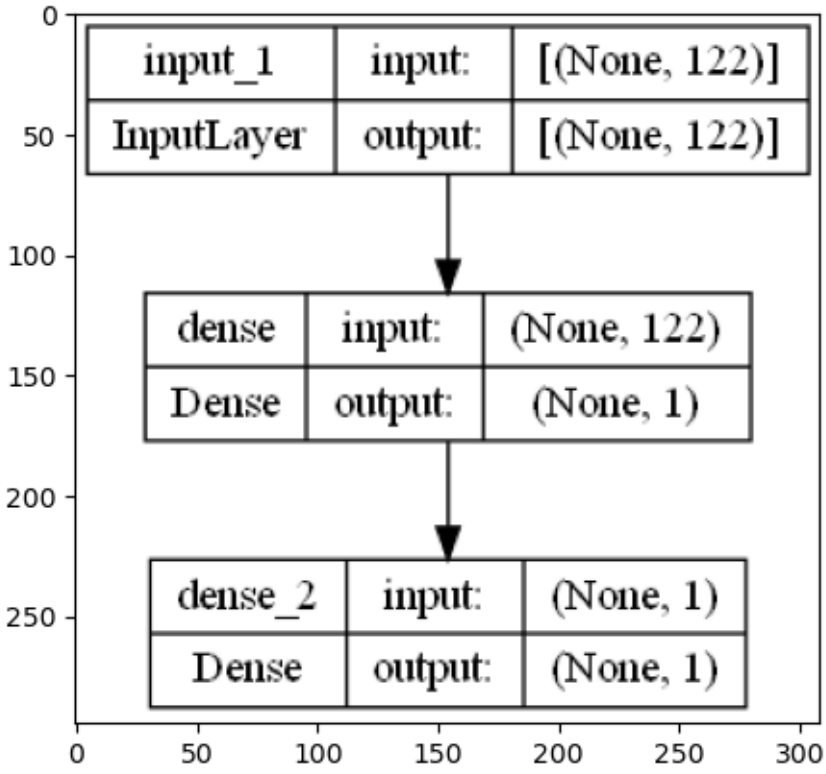

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

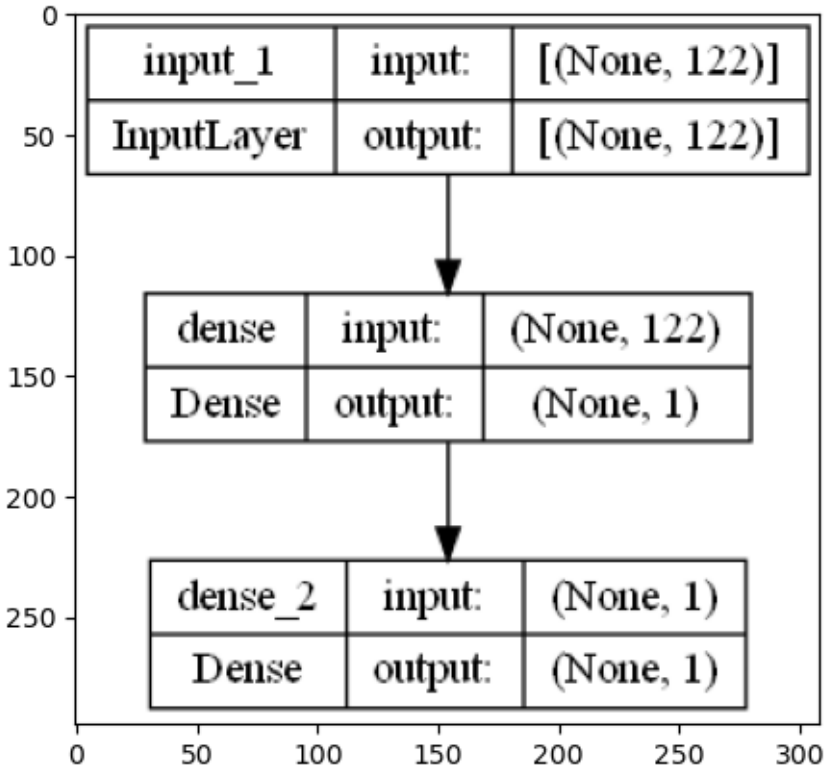

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

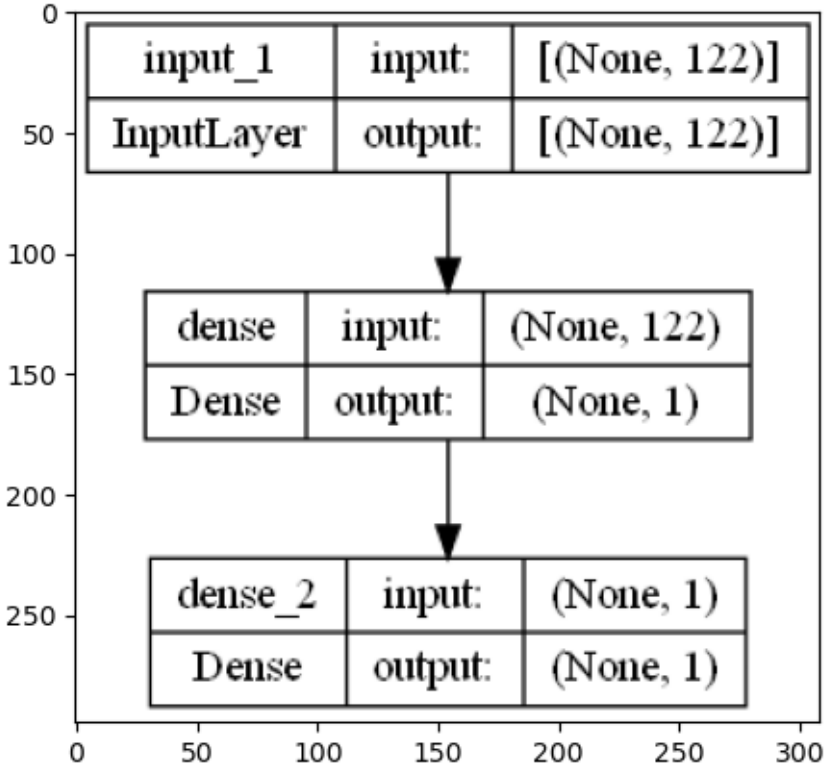

input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
300

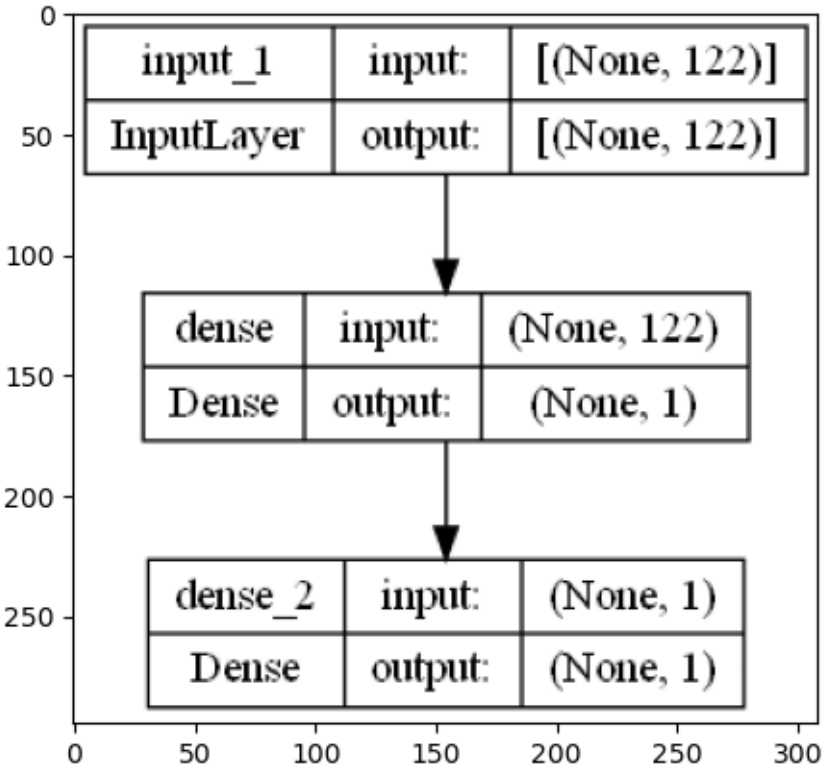

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

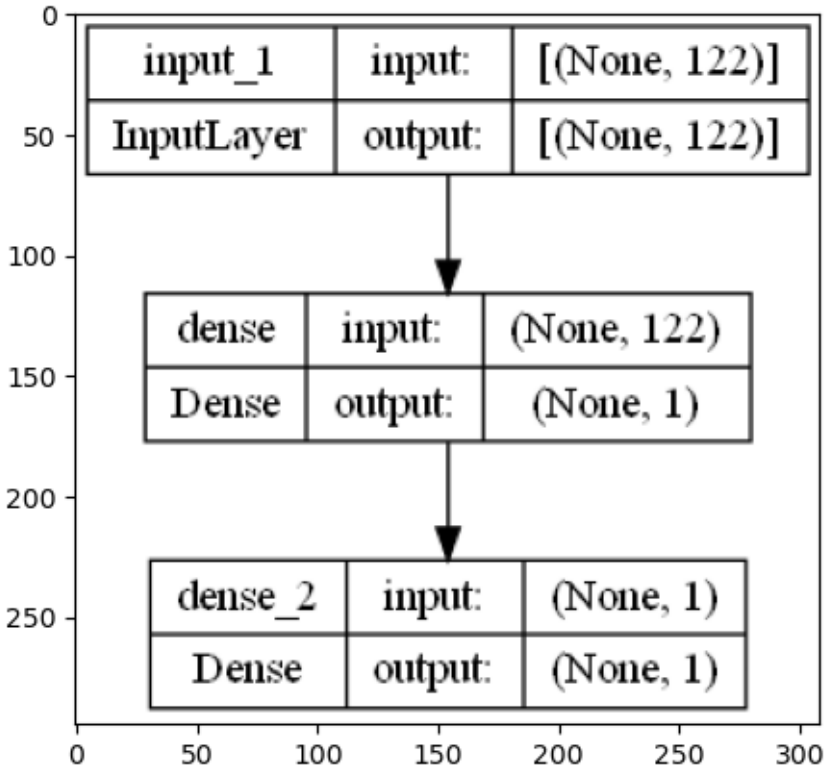

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

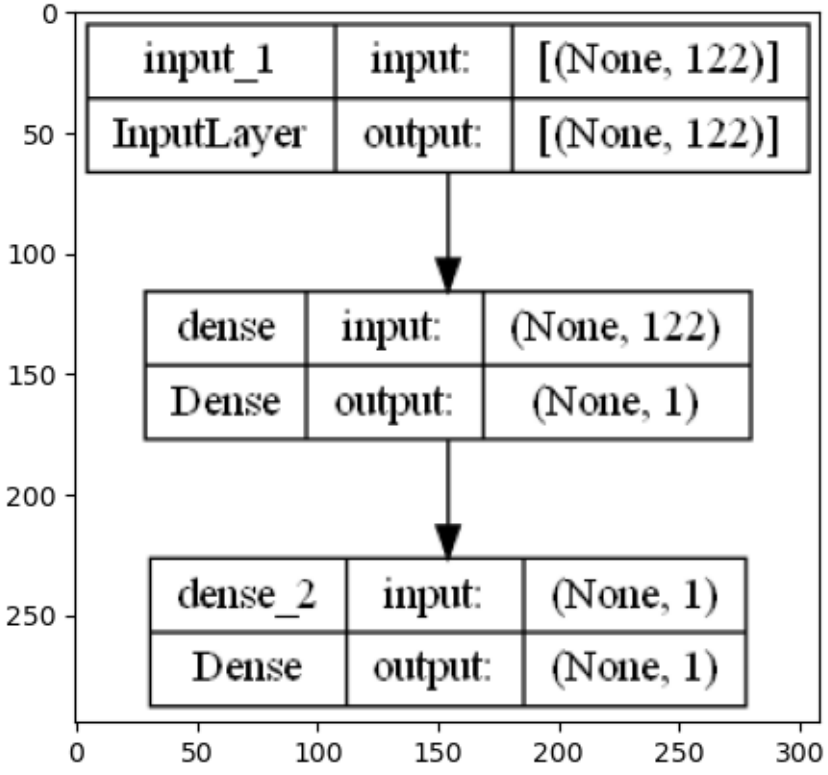

input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
0
50
100
150
200
250
300

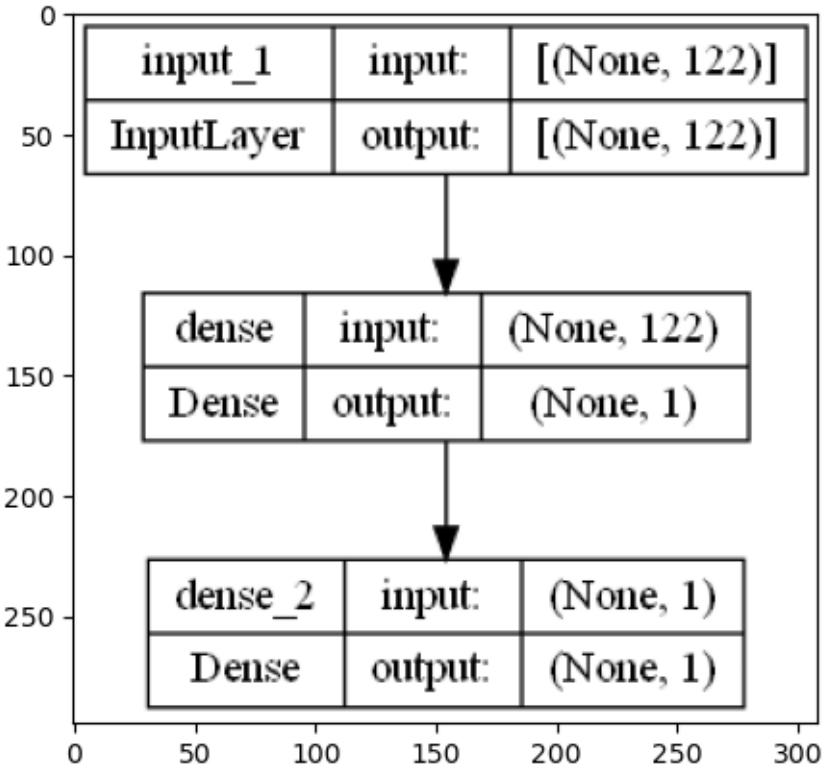

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

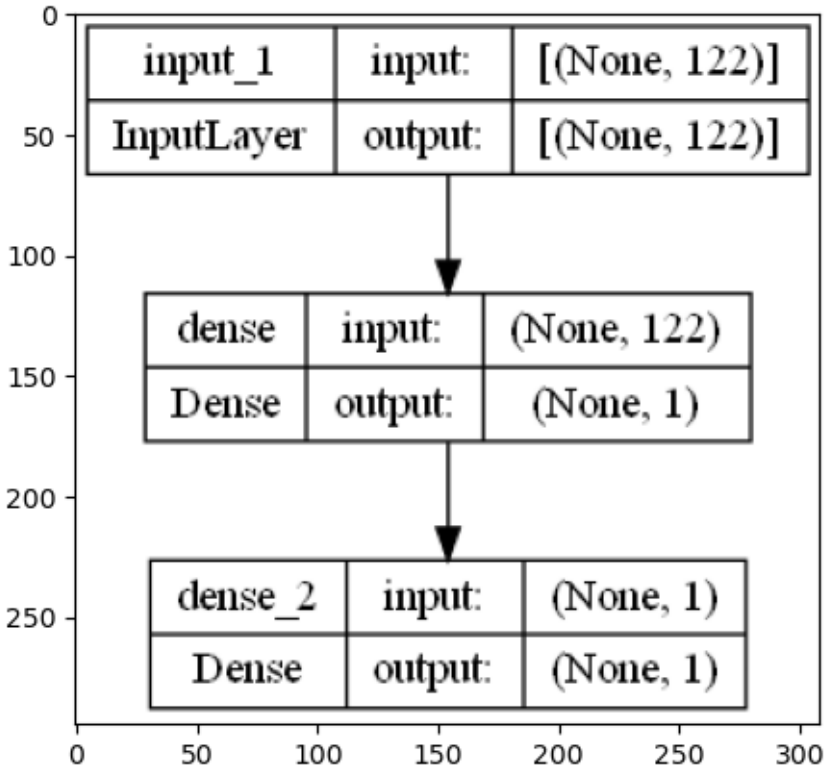

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

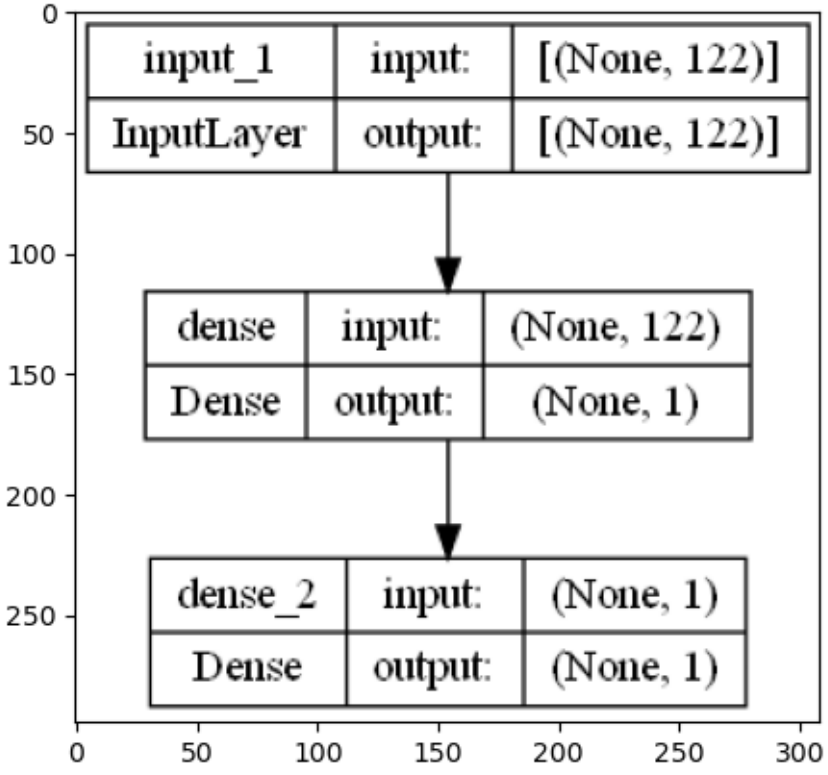

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

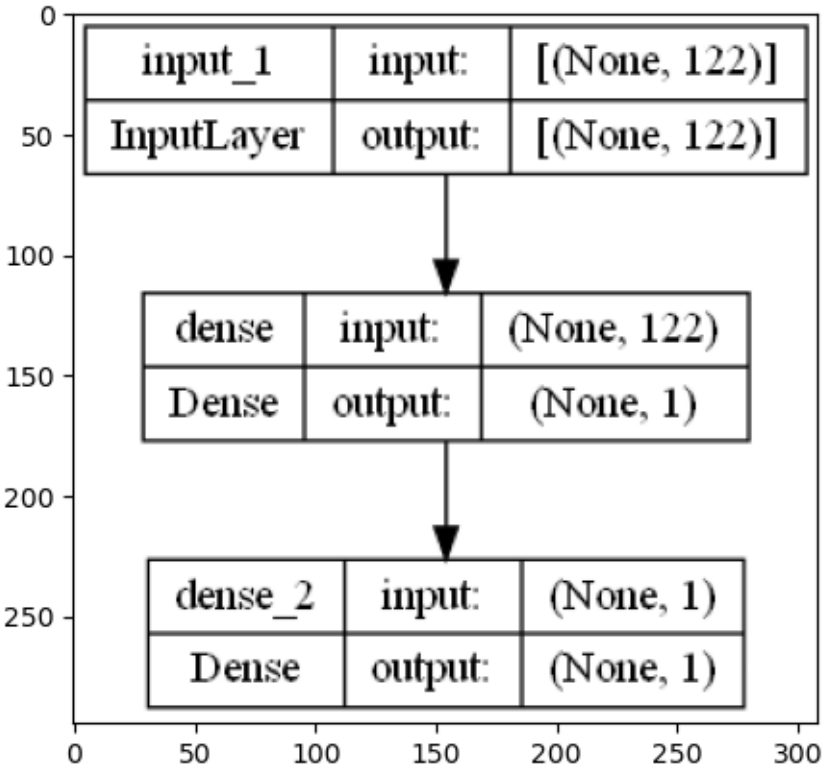

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

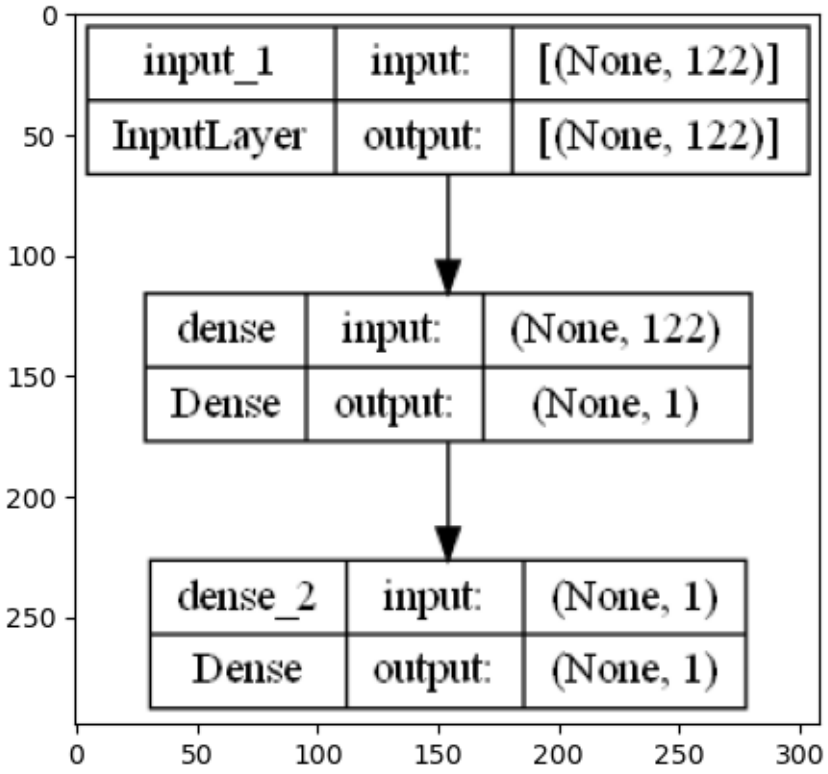

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

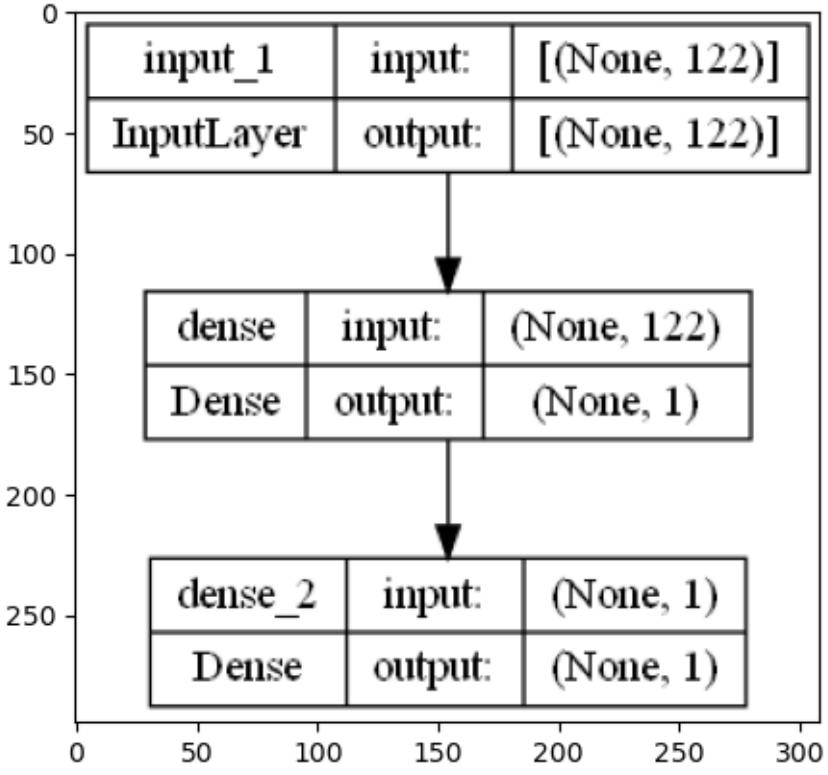

input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
0
50
100
150
200
250
300

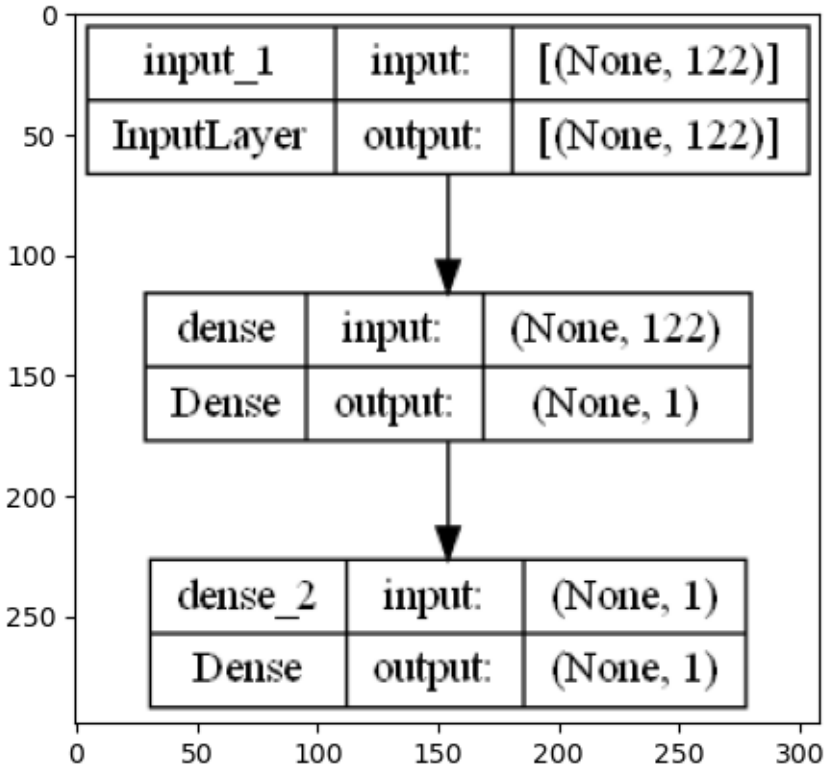

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

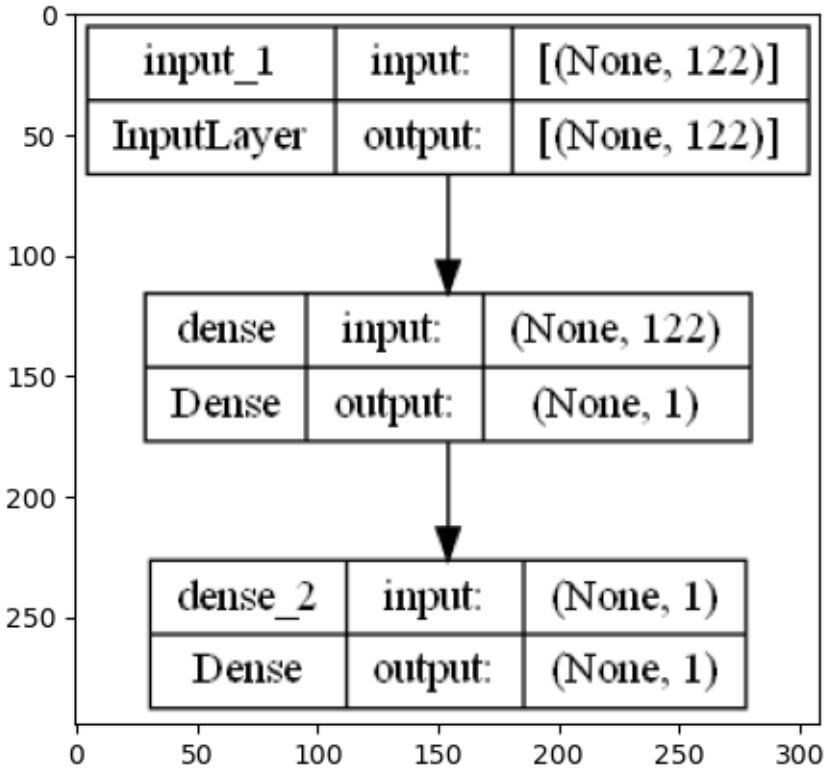

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

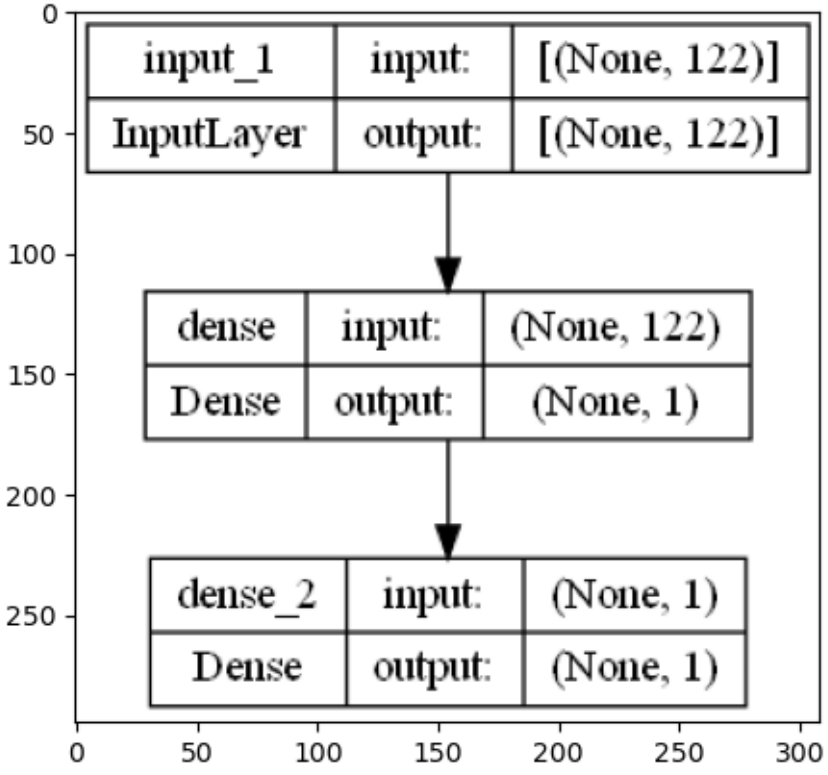

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

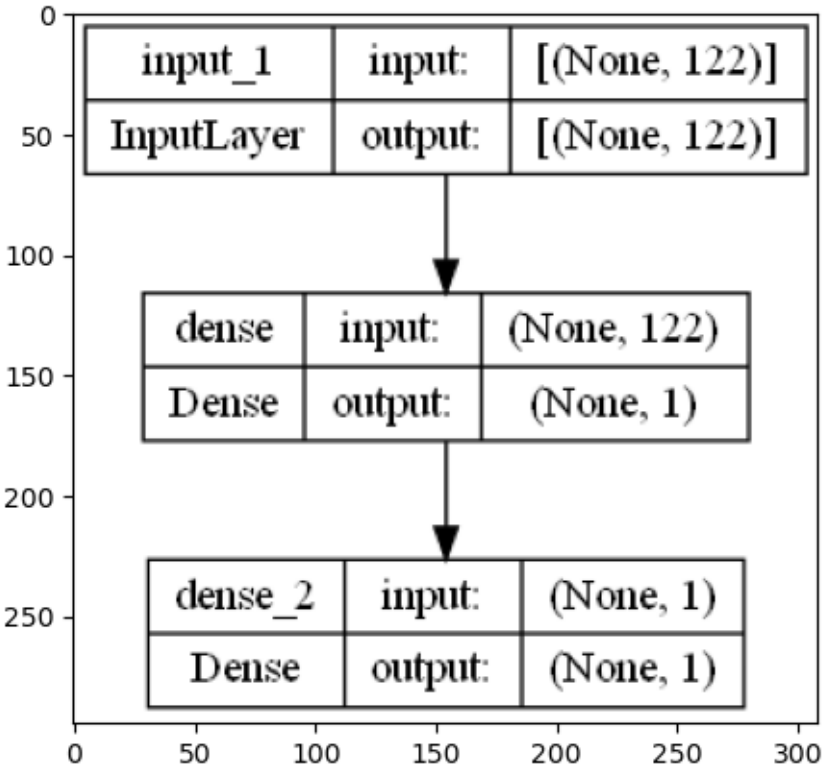

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

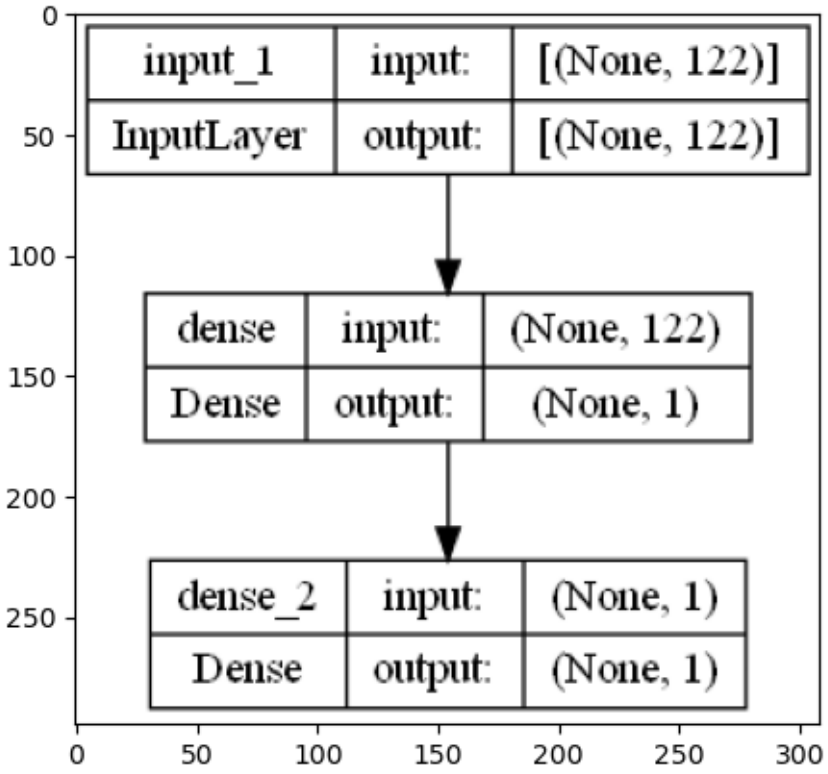

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

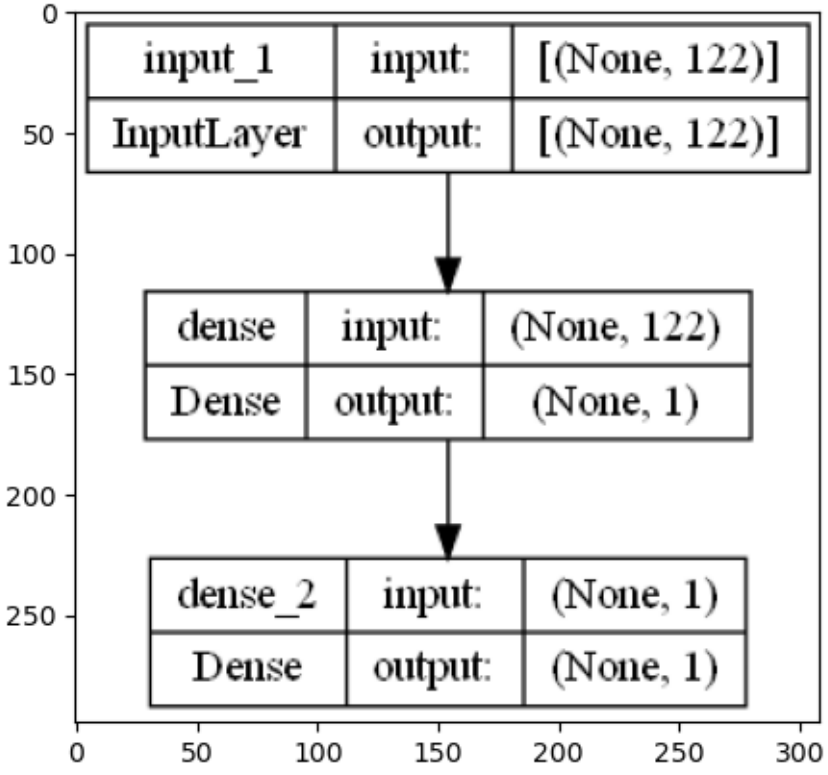

input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
300

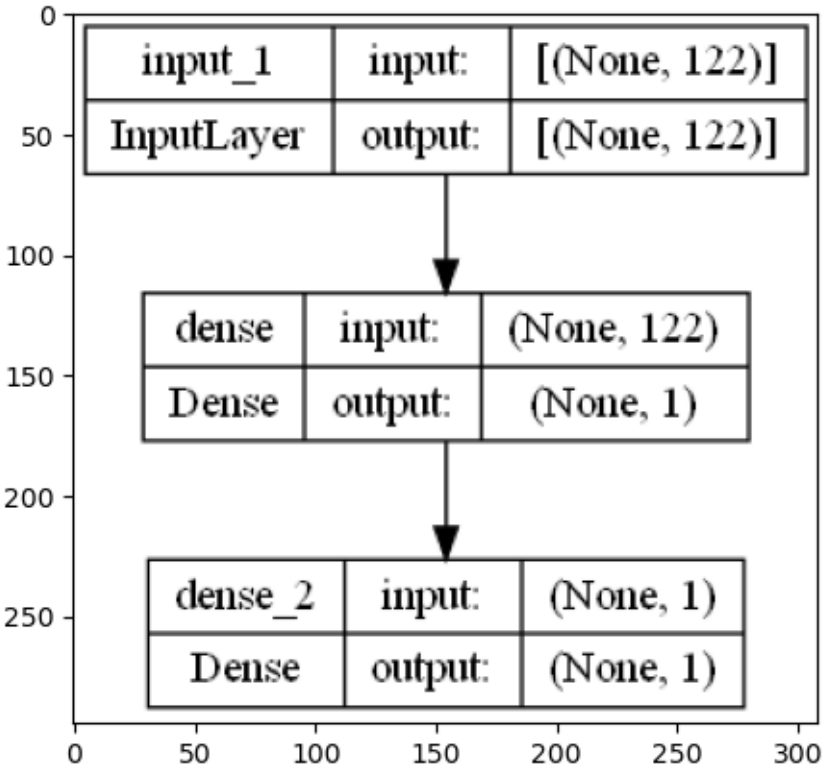

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

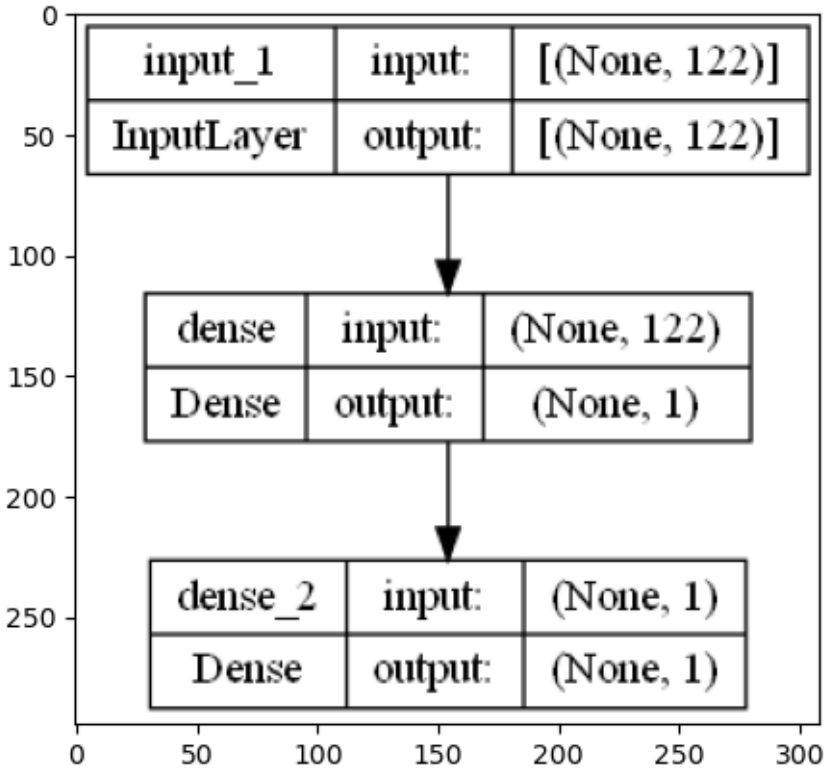

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

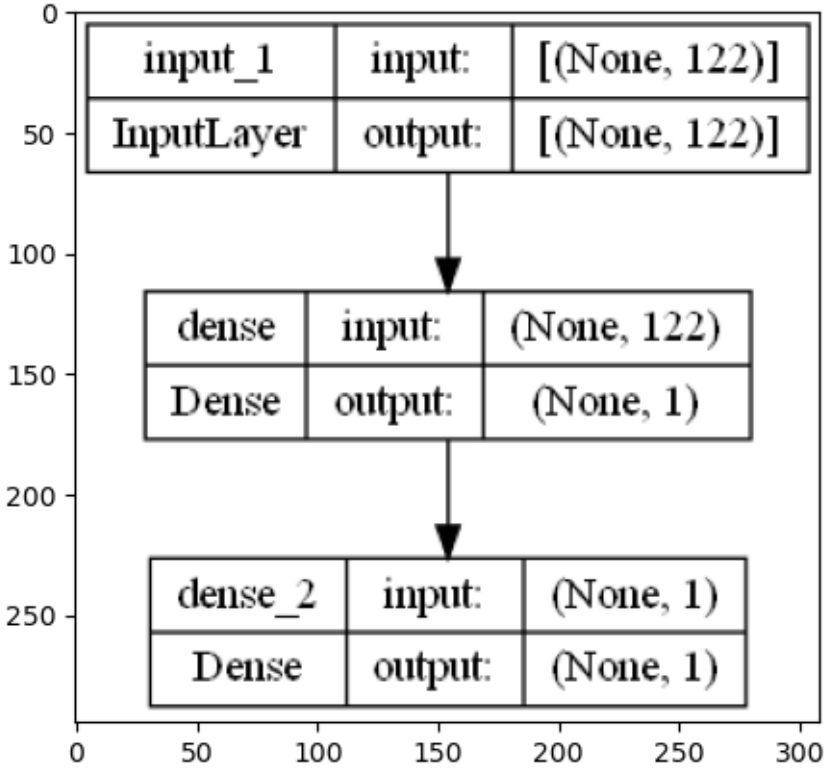

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300


input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

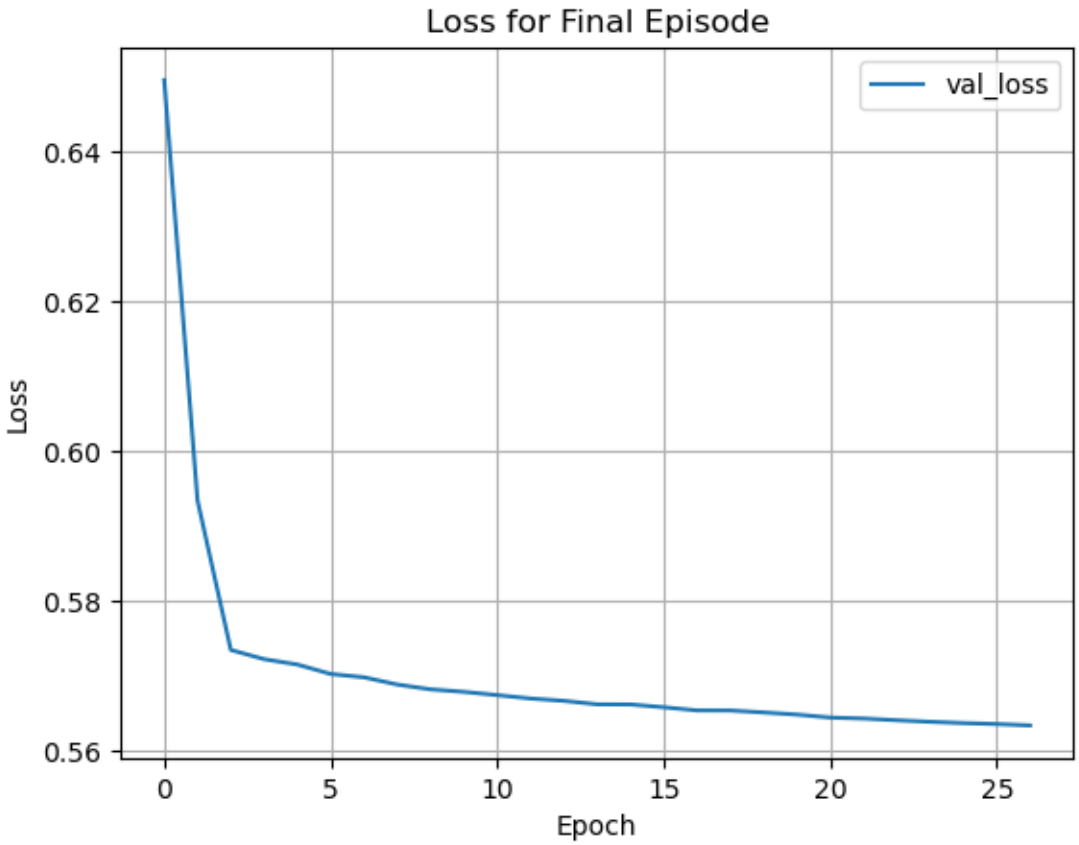

input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300


input_1
input:
[(None, 122)]
InputLayer
output:
[(None, 122)]
dense
input:
(None, 122)
Dense
output:
(None, 1)
dense_2
input:
(None, 1)
Dense
output:
(None, 1)
0
50
100
150
200
250
300


input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300


input_1 | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 1)
dense_2 | input: | (None, 1)
Dense | output: | (None, 1)
0
50
100
150
200
250
300

[30]:
```
# Call the 'plot_loss' function to create a graph showing how the loss changes␣
 ↪over time.
# 'history_list' contains the historical data of the loss values.
# "Loss for Final Episode" is the title that will be displayed at the top of␣
 ↪the graph.
plot_loss(history_list, "Loss for Final Episode")
```



#Implementing a Convex Neural Network with CVXPY

[31]:
```
# Generate toy data
# This line creates some fake data to play with. 'X' will be the input features␣
 ↪and 'y' will be the target labels.
# The data will have 1000 samples, each with 10 features. Out of these 10␣
 ↪features, 5 are actually informative.
# 'random_state' is like setting a seed so that the random data generated is␣
 ↪always the same.
```

```
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,␣
 ↪random_state=42)
# Preprocess data
# Standardizing the data by making it have a mean of 0 and a standard deviation␣
 ↪of 1.
# This can sometimes make it easier for machine learning models to learn from␣
 ↪the data.
X = StandardScaler().fit_transform(X)
# Split data into train and test sets
# This is dividing the data into two parts: one for training a machine learning␣
 ↪model, and
# the other for testing how well the model has learned.
# Here 80% of the data goes into training (X_train and y_train) and 20% goes␣
 ↪into testing (X_test and y_test).
# 'random_state' ensures that the split is reproducible.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# Here 'm' represents the number of training samples and 'n' represents the␣
 ↪number of features in each sample.
# This line is just storing these numbers for later use.
m, n = X_train.shape
```

#Hinge Loss as a Performance Metric for Binary Classification

[32]:
```
# 'theta' is a variable representing the weights in a linear model.
# 'n' represents the number of features in the data, so there is one weight for␣
 ↪each feature.
theta = cp.Variable(n)  # weight vector
# 'b' is another variable called bias. It's like an offset, deciding the␣
 ↪starting point of the decision boundary.
b = cp.Variable()  # bias term
# The 'hinge loss function' measures how well the model is doing, particularly␣
 ↪for classification tasks.
# 'cp.pos' is calculating the positive part of the expression in the␣
 ↪parentheses.
# If the expression inside is negative, it considers it as zero.
# Basically, this is summing up the errors made by the model on the training␣
 ↪data.
loss = cp.sum(cp.pos(1 - cp.multiply(y_train, X_train @ theta + b))) / m  #␣
 ↪hinge loss function
# 'L1 regularization' is a technique to prevent the model from fitting the␣
 ↪training data too perfectly,
# which can lead to bad performance on new, unseen data.
# It's doing this by trying to keep the weights small.
reg = cp.norm(theta, 1)  # L1 regularization
```

```
# 'lam' is the regularization strength. It determines how strongly we want to␣
 ↪regularize the model.
# A higher value means more regularization.
lam = 0.1
# 'objective' is what the model is trying to minimize.
# It's a combination of the hinge loss (how bad the model is doing on the␣
 ↪training data)
# plus the regularization term (keeping the weights small).
objective = cp.Minimize(loss + lam * reg)
# 'constraints' are conditions that the solution must satisfy.
# Here, the weights and bias must be greater than or equal to zero.
constraints = [theta >= 0, b >= 0]
# Here, we define the optimization problem by specifying the objective to be␣
 ↪minimized
# and the constraints to be satisfied.
problem = cp.Problem(objective, constraints)
# Now, we are actually solving the optimization problem.
# It finds the values for weights and bias that minimize the objective.
problem.solve(verbose=True)
# Getting the optimal weights and bias from the solved problem.
w = theta.value
b = b.value
# Evaluate model on test set
# We use the weights and bias to make predictions on the test data.
# '@' represents matrix multiplication of the test data and the weights, to␣
 ↪which the bias is added.
# 'np.sign' converts positive values to 1 and negative values to -1.
y_pred = np.sign(X_test @ w + b)

# Calculate the accuracy as the fraction of predictions that exactly match the␣
 ↪true target labels.
accuracy = np.mean(y_pred == y_test)
# Print the final results.
print("Optimum loss value is: ", problem.value)
print("Optimum weights values are: ", w)
print("Optimum bias value is: ", b)
```

```
===============================================================================
                                     CVXPY
                                     v1.3.1
===============================================================================
(CVXPY) Jun 24 07:31:23 PM: Your problem has 11 variables, 2 constraints, and 0
parameters.
(CVXPY) Jun 24 07:31:23 PM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Jun 24 07:31:23 PM: (If you need to solve this problem multiple times,
but with different data, consider using parameters.)
```

```
(CVXPY) Jun 24 07:31:23 PM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Jun 24 07:31:23 PM: Compiling problem (target solver=ECOS).
(CVXPY) Jun 24 07:31:23 PM: Reduction chain: Dcp2Cone -> CvxAttr2Constr ->
ConeMatrixStuffing -> ECOS
(CVXPY) Jun 24 07:31:23 PM: Applying reduction Dcp2Cone
(CVXPY) Jun 24 07:31:23 PM: Applying reduction CvxAttr2Constr
(CVXPY) Jun 24 07:31:23 PM: Applying reduction ConeMatrixStuffing
(CVXPY) Jun 24 07:31:23 PM: Applying reduction ECOS
(CVXPY) Jun 24 07:31:23 PM: Finished problem compilation (took 2.294e-02
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Jun 24 07:31:23 PM: Invoking solver ECOS  to obtain a solution.
-------------------------------------------------------------------------------
                                    Summary
-------------------------------------------------------------------------------
(CVXPY) Jun 24 07:31:23 PM: Problem status: optimal
(CVXPY) Jun 24 07:31:23 PM: Optimal value: 4.813e-01
(CVXPY) Jun 24 07:31:23 PM: Compilation took 2.294e-02 seconds
(CVXPY) Jun 24 07:31:23 PM: Solver (including time spent in interface) took
4.986e-03 seconds
Optimum loss value is:  0.4812500000008167
Optimum weights values are:  [2.64627883e-13 4.79997865e-13 4.44370865e-13
1.94254214e-12
 9.72147814e-13 1.07267438e-12 5.79441265e-15 3.30249998e-13
 5.83954547e-13 2.07067613e-12]
Optimum biases values are:  1.3701231433637668
```

1.0.11 Results from Descriptive Data Analysis (300-350 words):

In the following cell, describe how the results obtained from the descriptive analysis can help answer the chosen business problem or provide insight into the issue. Provide a clear and concise response that refers to the analysis results. [15 Marks]

5. Descriptive Data Analysis Results Interpretation The descriptive analysis of the NSLKDD dataset provides valuable insights into the nature of network connections, which can be used to address network security concerns. The pie charts for the 'protocol_type' and 'final results' variables supply us with a clear image of the distribution of different styles of protocols used in the network connections and the outcomes of these connections, respectively. In order to identify a network's normal behaviour, it is helpful to know in what proportions 'tcp', 'udp', and 'icmp' are used. There may be some indication of unusual network activity or a possible cyber threat if there is any significant change from this distribution. For example, an increase in 'icmp' traffic for no reason can indicate a denial-of-service attack or scanning attack on the

network [5]. There are several types of network outcomes depicted in the 'outcome' pie chart. It is particularly useful for understanding what types of attacks a network is vulnerable to. As an example, if a large proportion of outcome types are 'neptune' or 'smurf' attacks, this indicates the network is frequently attacked by denial-of-service attacks [6]. In this case, the 'protocol_type' pie chart shows that almost all of the network connections use the 'icmp' protocol, followed by way of 'tcp' and 'udp'. This data is important because it enables us to recognise the most generally used protocols in network connections. It can help network administrators to focus their efforts on securing these protocols, as they may be, likely to be the most targeted due to their incidence. The 'outcome' pie chart shows that a significant portion of the network connections are getting attacked. This is a clear indication of the severity and frequency of network attacks, emphasising the need for robust network intrusion detection systems. Based on this information, the cybersecurity team can prioritise defences and resources to mitigate specific threats.

1.0.12 Results from Exploratory Data Analysis (300-350 words):

In the following cell, describe how the results obtained from the exploratory analysis can help answer the chosen business problem or provide insight into the issue. Interpret the graphs generated and provide a clear and concise response that refers to the plots. Your answer should include at least two graphs. [15 Marks]

6. Exploratory Data Analysis Results Interpretation The exploratory data analysis, particularly the "Loss for Final Episode" graph, provides insights into the performance of the machine learning model used for anomaly detection. This graph plots the model's loss on both the training and validation data over multiple epochs of training. The loss function is a measure of the model's prediction error, with a lower loss indicating better model performance. The decreasing trend of loss over time suggests that the model is learning from the training data and improving its ability to predict network outcomes. This is crucial for maintaining network security as a more accurate model can provide timely and reliable detection of network anomalies or attacks. For instance, a model with a low loss might be able to accurately detect and alert cybersecurity teams of potential threats, allowing them to respond quickly and mitigate any damage. However, it's also important to monitor the model's performance on the validation data, which is unseen data not used in training. The decreasing loss on validation data suggests that the model is fitting and is not overfitting or underfitting to the training data so that it is able to generalise well to new data. Moreover, it shows that at around 0.25 epoch, the validation loss reduced reasonably well, and the model's performance became very stable. This is crucial for making sure of the model's robustness and reliability in actual-world applications [7]. As a result, the descriptive and exploratory analyses provided valuable insights that can be used to improve anomaly detection models, thereby enhancing network security and safeguarding the network against cyber threats.

1.0.13 References

Please add your references in the following cell and ensure to accurately cite them using the IEEE referencing style in your response. You can find more information about the IEEE referencing style in this link. https://www.bath.ac.uk/publications/library-guides-to-citing-referencing/attachments/ieee-style-guide.pdf

[1] P. Sharma and C. K.Reddy, “A detailed analysis on intrusion detection datasets”, In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 3398-3399.

[2] M.Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set”, In: Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009, pp. 1-6.

[3] unb, “NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” https://www.unb.ca/cic/datasets/nsl.html

[4] “NSL-KDD,” Kaggle, Apr. 25, 2019. https://www.kaggle.com/datasets/hassan06/nslkdd

[5] D. Rafter, “What are Denial of Service (DoS) attacks? DoS attacks explained”, 2022. https://us.norton.com/blog/emerging-threats/dos-attacks-explained

[6] K. Labib, and V. R.Vemuri, “Detecting And Visualizing Denial-of-Service and Network Probe Attacks Using Principal Component Analysis”, 2005. http://www.cs.ucdavis.edu/~vemuri/papers/pcaDos.pdf

[7] J. Won, J.-W. Park, S. Jang, Jin, and Y. Kim, “Automated Structural Damage Identification Using Data Normalization and 1-Dimensional Convolutional Neural Network”, Appl. Sci. 11, pp. 2610, 2021. https://doi.org/10.3390/app110626102021