Journal of Ambient Intelligence and Humanized Computing (2022) 13:3457–3466
https://doi.org/10.1007/s12652-020-01848-9

ORIGINAL RESEARCH

# Deep learning-based classification model for botnet attack detection

**Abdulghani Ali Ahmed**[1] · **Waheb A. Jabbar**[2] · **Ali Safaa Sadiq**[3] · **Hiran Patel**[3]

Received: 19 October 2019 / Accepted: 26 February 2020 / Published online: 9 March 2020


**Abstract**
Botnets are vectors through which hackers can seize control of multiple systems and conduct malicious activities. Researchers have proposed multiple solutions to detect and identify botnets in real time. However, these proposed solutions have difficulties in keeping pace with the rapid evolution of botnets. This paper proposes a model for detecting botnets using deep learning to identify zero-day botnet attacks in real time. The proposed model is trained and evaluated on a CTU-13 dataset with multiple neural network designs and hidden layers. Results demonstrate that the deep-learning artificial neural network model can accurately and efficiently identify botnets.

**Keywords** Security · Botnet · Feed-forward · Artificial neural network · Backpropagation · Deep learning

## 1 Introduction

In our modern world, computers have become a convenient and ubiquitous part of our everyday lives. However, the increased proliferation of computer systems has introduced new kinds of security risks. These security risks can compromise user data or severely hamper the operations of computer systems, potentially causing complete system failures (Shah et al. 2013).

Botnet detection has become a popular subject in the cybersecurity literature. Botnets are a type of network-based attack that seeks to subvert multiple computers simultaneously and turn them into "zombie" systems, as shown in Fig. 1. These "zombie" computers are then used for malicious activities such as identifying theft, distributed denial of service attacks (DDoS), phishing, spamming, and domain name system spoofing.

This paper reviews several methods in the literature for detecting botnet attacks. Numerous studies in cybersecurity literature (Ahmed 2015; Ahmed et al. 2013a, b, 2016) have covered such attacks. These studies have employed machine-learning algorithms such as support vector machine (SVM) (Narang et al. 2014), decision tree (Dai et al. 2016), naïve Bayes (NB) (Kalaivani and Vijaya 2016), bees (Jantan and Ahmed 2014a, b) and random forest (Singh et al. 2014). However, a topic that has been rarely covered is the use of deep learning algorithm (Svozil et al. 1997) for training artificial neural networks (ANNs) to detect botnets.

This paper makes two significant contributions to the literature. First, it evaluates the efficiency and accuracy of a deep neural network (DNN) for botnet attack detection. Second, a DNN algorithm is used on a CTU-13 dataset (Garcia et al. 2014) with multiple neural network (NN) designs and hidden layers to determine the abilities of the proposed technique. The rest of this paper is arranged as follows: Sect. 2 covers the literature review. Section 3 covers the feed-forward backpropagation ANN technique. Section 4 discusses the implementation of the proposed model. Section 5 contains the results and analysis. Section 6 presents the conclusions and provides ideas for future research.

✉ Abdulghani Ali Ahmed
abdulghani@ump.edu.my

Waheb A. Jabbar
waheb@ieee.org

Ali Safaa Sadiq
ali.sadiq@wlv.ac.uk

Hiran Patel
H.Patel10@wlv.ac.uk

[1] Safecyber Systems Corporation, 26300 Kuantan, Pahang, Malaysia

[2] Faculty of Electrical and Electronics Engineering Technology, Universiti Malaysia Pahang, 26600 Pekan, Pahang, Malaysia

[3] School of Mathematics and Computer Science, University of Wolverhampton, Wulfruna Street, Wolverhampton WV1 1LY, UK

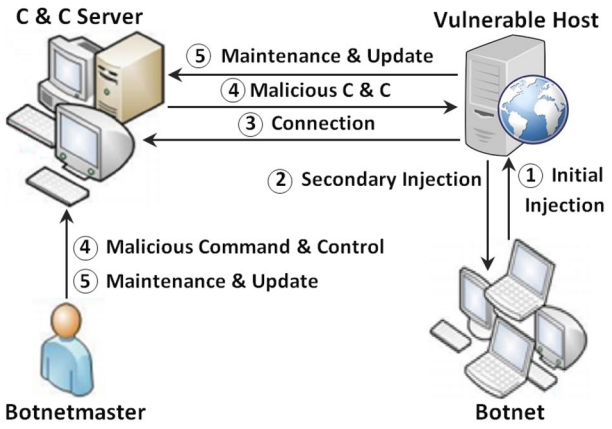


**Fig. 1** Typical botnet life cycle

## 2 Literature review

Multiple methods have been proposed in the literature to identify botnets. One study (Karasaridis et al. 2007) has used an anomaly-based botnet detection method to identify botnet controllers using transport layer data, thus enabling the detection of IRC botnet controllers without known signatures or captured binaries and making it a passive method that is invisible to operators, scale to large networks, and protects end users. This method also determines a botnet's size and activities from outside of compromised networks, making it capable of identifying botnets using encrypted and obfuscated protocols.

A Botsniffer (Gu et al. 2008) that uses network-based anomaly detection was developed to locate and identify botnet command and control (C&C) channels in local area networks without the use of botnet signatures. This technique relies on the tendencies for parts of a botnet to possess similar spatiotemporal correlations and behaviors because of preprogrammed activities related to C&C communication in the protocol layer. The Botsniffer uses statistical and analysis algorithms to track botnets with centralized IRC architectures and network traffic with a low false positive rate, respectively.

A BotDigger (Al-Duwairi and Al-Ebbini 2010) was developed to detect botnets using the logical rules and features that define their behavior. The BotDigger measures the influence of fuzzy member sets to infer human reasoning and decision-making. All techniques that employ fuzzy logic, amount, type, and behavior of fuzzy member functions and rules exert a substantial influence.

A host-based botnet detection (Masud et al. 2008) that correlates multiple log fields using a flow-based detection method was developed to segregate Botmaster commands into different categories. Bots have faster reaction times than humans, thus simplifying the mining or correlation of multiple log files. This technique efficiently identifies certain kinds of C&C traffic by correlating multiple host-based log files from IRC bots. The technique also works on non-IRC bots because it can detect C&C traffic before a payload is identified.

Several studies have used machine-learning techniques to identify botnets, and the decision tree (Dai et al. 2016) method is popular for differentiating between botnet and non-botnet traffic. This technique abstracts classification rules into decision tress using disordered and irregular instance groups. This technique compares internal decision tree node qualities, values downward branches according to node attributes, and derives conclusion using leaf nodes in a top–down recursive manner. This technique ensures that root-to-leaf nodes represent conjunctive nodes, and the entire tree represents groups of disjunctive expression rules. This technique has advantages because the decision tree classification algorithm creates rules that are easy to understand for different data types without using large amounts of computational resources. The decision tree can identify the significance and limitations of certain nodes (such as difficulties in estimating continuation fields) through the extensive pre-treatment of chronological data. Decision trees may suffer from errors when using numerous categories.

An NB (Kalaivani and Vijaya 2016) classifier proposed to process natural language and retrieve information is a simple and effective method using Bayesian theorems. This classifier is suitable for inputs with large amounts of dimensionality. This classifier assumes that the variables' effect on a given class are not influenced by the values of other variables. For example, the NB inducer derives class conditional probabilities to identify the one with the uppermost posterior. The NB classifier can be used as a supervised machine-learning algorithm for certain probability models.

SVM (Narang et al. 2014) is a supervised pattern classification method developed for pattern recognition. This algorithm uses machine learning to derive training classification and regression rules. This algorithm efficiently handles high-dimensionality feature spaces owing to its solid mathematical foundation, and it can provide simple and effective results.

Existing studies are helpful (Cui et al. 2018) but demonstrate slow speed and poor accuracy in detecting malware. The DNN approach was recently introduced as an efficient method to detect malware. The key point of DNN methods is their capability in achieving a high detection rate while generating a low false-positive rate. DNN-based studies(Cui et al. 2018; Ye et al. 2018; Kolosnjaji et al. 2016; Saxe and Berlin 2015; Vinayakumar et al. 2017) have demonstrated promising results in identifying malicious code variants, detecting intelligent malware, classifying malware system call sequences, and detecting and classifying Android malware. This paper uses a

deep learning ANN model to train NNs for botnet attack detection. The developed model is compared with other machine learning-based algorithms to determine its efficiency and effectiveness.

On the other hands, the authors in Al Shorman et al. (2019) have introduced new unsupervised evolutionary Internet of Things (IoT) based botnet detection method. The foremost goal of their proposed method was to distinguish IoT botnet attacks that triggered form compromised IoT devices. They have achieved this by take advantage of the efficiency of the modern swarm intelligence algorithm known as Grey Wolf Optimization algorithm (GWO). GWO was used to optimize the hyperparameters of their baseline One Class Support Vector Machine (OCSVM). Their model was also tending to find the features that best describe the IoT botnet problem. This paper uses a deep learning ANN model to train NNs for botnet attack detection. In another attempt, to produce a new android dataset, Moodi and Ghazvini (2019) have come out with 28 Standard Android Botnet Dataset (28-SABD). They have used ensemble K-Nearest Neighbors (KNN) technique as a way to advance the accuracy of the allocated labels by the signature-based method. However, the obtained overall accuracy was 94%, which indicates that there is still need for further improvement or need for more accurate detection model. In contrast, Wang et al. (2019) have tried to reduce the false positives of DDoS attacks by cultivating execution efficiency and improving the relationship between detection and prevention courses. In their work, a defensive mechanism based on honeynet technology was introduced. Yet, these types of technologies are analytically expensive and they enquire more knowledge to be feeding into the model for better detection with more dynamic data behavior, where deep-learning based model is needed to overcome such limitation Maimó et al. (2019). While a novel multi feature behavior approximation algorithm was proposed by Dhaya and Ravi (2020) as a way to increase the performance of botnet detection. A multi feature behavior approximation algorithm was introduced to monitor each transaction performed by different users. However, there is always still room for improvements to have more robust detection model to advance this research area. Hence, taking the advantages that introduced by the recent introduced deep learning models, and to overcome the aforementioned limitations, this paper uses a deep learning ANN model to train NNs for botnet attack detection. The developed model is compared with other machine learning-based algorithms to determine its efficiency and effectiveness.

## 3 Botnet detection

This section studies the detection of Botnet using two main parts: machine learning and deep learning. For the first part, a feed-forward backpropagation ANN is presented as a preliminary study to show the efficiency of using DNN model in detecting Botnet attacks compared with machine learning techniques. The second part studies the deep-learning model for the detection of Botnet attack. These two parts are further explained in the following subsections.

### 3.1 Feed-forward backpropagation ANN technique

The feed-forward backpropagation ANN technique has two main components, namely, preprocessing and classifier, as shown in Fig. 2.

Network traffic is analyzed on a flow level by the preprocessing component, which extracts a set of features for all traffic flows. The selection of features that effectively identify botnet attacks is critical in flow-based traffic analysis. A total of 15 traffic flow statistical features are extracted, as shown in Table 1.

Common features alone are not sufficient to differentiate botnet traffic from normal traffic (Kalaivani and Vijaya 2016). Hence, this paper employs new features, such as average byte rate, average packet rate, ping bytes, time comparison, and malicious ports, to identify botnet activities. The development of a botnet/non-botnet traffic model is conducted by the classifier component using information from the preprocessing component. The classification process has two phases: training and testing. The backpropagation learning algorithm is selected, and data are represented in the training phase. This information is used to map inputs to desired outputs. The feed-forward and backward processes (Rumelhart et al. 1995) are used to train the developed model to predict the outputs of certain inputs, as illustrated in Fig. 3.

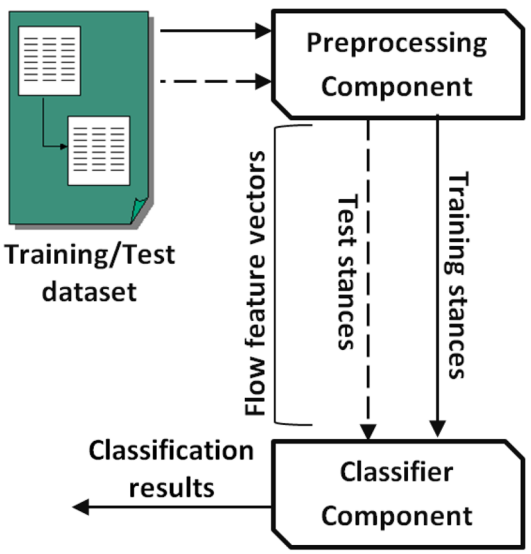


**Fig. 2** Feed-forward backpropagation technique for botnet detection

**Table 1** List of extracted traffic flow features

| No | Feature | Description |
|---|---|---|
| 1 | Start time | Beginning of traffic flow |
| 2 | Duration | Total time taken for a particular flow to complete |
| 3 | Protocol | Use of a TCP, ICMP, UDP, or SMTP protocol |
| 4 | Source and destination IP addresses | Origin and destination of a packet |
| 5 | Direction | Path taken by a packet |
| 6 | Source and destination ports | Data service or location where a request should be sent |
| 7 | State | A SYN, RST, CON, ACK, or FIN flow state |
| 8 | Type of service (ToS) | Specific treatment or priority of each IP packet |
| 9 | Total packets | Number of packets in a specific flow or number of packets transmitted within a specific flow/time |
| 10 | Total bytes | Total number of bytes that the client sends for each request |
| 11 | Time comparison | Comparison between flow start time and flow end time |
| 12 | Average byte rate | Average byte rate calculated using total bytes and duration |
| 13 | Average packet rate | Average packet rate calculated using total packets and duration |
| 14 | Ping byte | A packet with a size larger than 65,435 bytes is considered malicious |
| 15 | Malicious port | Port number is also used to obtain information on remote systems that may be the target of malicious attacks |

For a feed-forward back propagation ANN-based network with $x_n$ input (i) nodes, h hidden (j) nodes, and o output (k) nodes, the back propagation training cycle has a forward and backward phase. In the forward phase, a set of input vectors $x_1,\ldots, x_n$ is propagated through multiplication with associated weights $w_1,\ldots, w_n$. Prior node outputs are multiplied with their respective weights and summed to calculate the net input for the jth node in the hidden layer as follows:

$$net_j = \sum_{i=1}^{n} w_{ji}\, x_i \tag{1}$$

The value obtained from (1) specifies the ANN neuron outputs, which become input values for neurons in the next linked layer. Thus, the output (activation) of the jth node in the hidden layer is provided by the following:

$$o_{hj} = \int (net_j). \tag{2}$$

The net input to the kth output node is calculated as follows:

$$net_k = \sum_{j=1}^{L} w_{kj}\, o_{hj}\,. \tag{3}$$

The net output $\omega_j$ to the kth output node is calculated as follows:

$$\omega_j = \int (net_k). \tag{4}$$

The error signal is propagated through the network in a backward direction to adjust weights and bias values throughout the backward phase. These calculated weight changes are then applied to free network parameters. During subsequent iterations, the entire process is repeated using the next training model to minimize statistical errors. The delta term for each output node $\varepsilon_k$ is provided by calculating the error signal for each output node $\Delta_{ok}$ (the difference between the targeted value $\varpi_k$ and the actual values $\omega_k$ in the output layer) and multiplying it by the actual output of that node multiplied by (1—its actual output).

$$\begin{aligned} \Delta_{ok} &= (\varpi_k - \omega_k\,), \\ \varepsilon_{ok} &= \Delta_{ok}\omega_k(1-\omega_k), \\ \varepsilon_{ok} &= (\varpi_k - \omega_k)\omega_k(1-\omega_k). \end{aligned} \tag{5}$$

The sum of the output node deltas for a particular hidden node are multiplied by the weight between that output and the hidden nodes to calculate the error signal for each hidden node $\Delta_{hj}$.

$$\Delta_{hj} = \sum_{k=1}^{w} \varepsilon_{ok}\, w_{kj}\,.$$

The error signal for the jth hidden node is then multiplied by its output and by (1—its output) to obtain the delta term for the jth hidden node $\varepsilon_{hj}$,

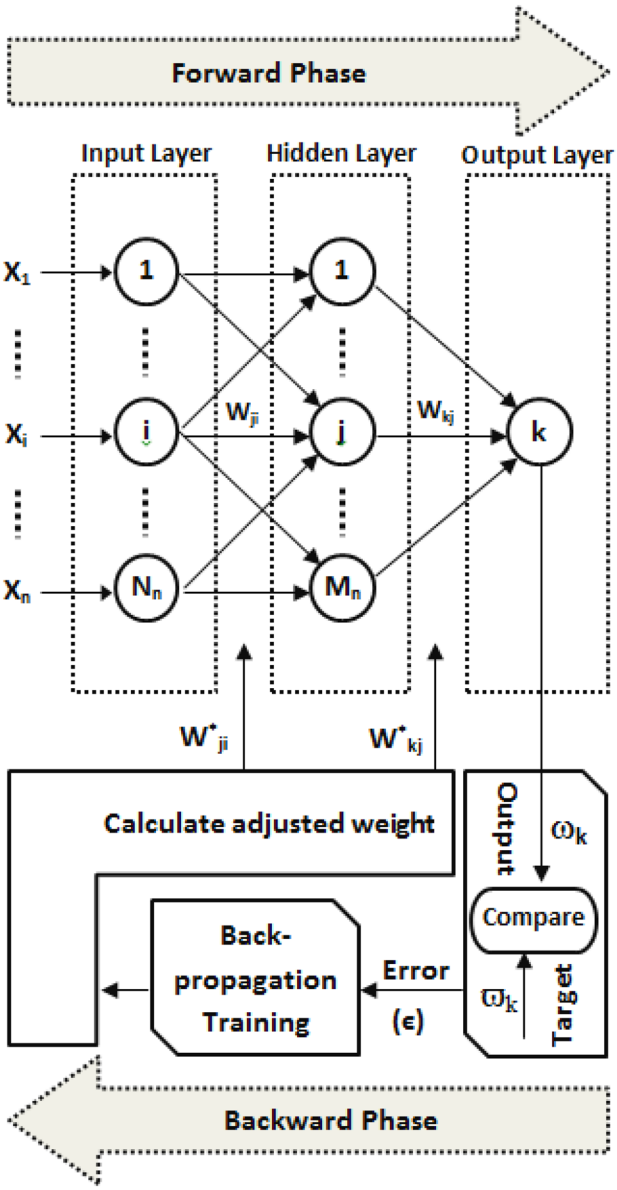


**Fig. 3** Methodology of proposed technique

$$\varepsilon_{hj} = \left(o_{hj}\right)\left(1 - o_{hj}\right) \sum_{k=1}^{W} \varepsilon_{ok}\, w_{kj}\,. \tag{6}$$

The delta of each output node is multiplied by the output (activation) of the hidden node to which it is connected to derive the weight error for each weight vector between the hidden and output nodes $\gamma_{jk}$. $\gamma_{jk}$ is used to adapt the weights between the output and the hidden layers as below.

$$\gamma_{jk} = \varepsilon_{ok}(o_{hj}). \tag{7}$$

The weight error derivatives for each weight between the input node and the hidden node $\gamma_{ij}$ are provided by multiplying the delta of each hidden node with the activation of the input node to which it is linked. $\gamma_{ij}$ is used to adapt the weights between the input and hidden layers as follows:

$$\gamma_{ij} = \varepsilon_{hj}(x_i).$$

A learning rate parameter σ is needed to perform changes on the weights themselves to update weights during each backpropagation cycle. Weights that link the hidden and output nodes at time (t + 1) are provided using the weights at time (t) and $\gamma_{jk}$ using the following equation:

$$w_{jk}(t+1) = w_{jk}(t) + \sigma\left(\gamma_{jk}\right). \tag{8}$$

Likewise, the weights that link input and hidden units are provided using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \sigma\left(wed_{ij}\right). \tag{9}$$

This equation ensures that every node in the ANN receives an error signal that shows its proportional contribution to the total errors between targeted and actual outputs. The update process for the weights that link nodes between layers depends on the error signal obtained by the nodes. The mean square error between the actual output of the ANN and its desired output is reduced for all sets of training inputs by iterating the two processes in (8, 9) for different input patterns and targets.

**Table 2** Optimizer parameter setup

| Parameter | Value |
|---|---|
| learning_rate | 0.001 |
| beta_1 | 0.9 |
| beta_2 | 0.999 |
| Epsilon | 1e−07 |
| amsgrad | False |

## 3.2 Deep-learning model for the detection of botnet

In the part of deep-learning, model is developed based on the Tenserflow platform using Adam (Kingma and Ba 2014) as an optimization algorithm for the first-order gradient-based optimization of stochastic objective function, which is obtaining the maximum accuracy of the classification rate for the botnet detection model. This optimizer works on adaptive estimates of lower-order instants. Using this method enables our developed model with an upfront implementation process. It is computationally efficient and has slight memory desires. Moreover, Adam's optimizer resists with slanting rescaling of the gradients of the problem space, and it is well-fitted for botnet detection problem with

immense volume in terms of data and/or features of attackers. Table 2 lists the main setup of Adam's optimization algorithm. The algorithm exponentially updates its moving averages of the gradient (mt) and the squared gradient β_1, β _2 ∈ [0, 1], regulating the exponential degeneration rates of the moving averages toward the optimal decision (in our case is the class 0/1 attack/non-attack). Optimization algorithm for the DNN-Botnet detection model is shown in Fig. 4.

The optimization parameters are defined as follows:

- learning_rate: A tensor or a floating point value, which indicates the learning rate.
- beta_1: The float value or a constant float tensor. The exponential degeneration rate for the 1st moment guesses.
- beta_2: The float value or a constant float tensor. The exponential degeneration rate for the 1st moment guesses.
- epsilon: A tiny constant for numerical stability of the model.
- amsgrad: Boolean value, which indicates whether to apply AMSGrad variant of this algorithm. The reader may refer to[24] for more details.

## 4 Implementation

This section covers the use of the proposed deep learning DNN and feed-forward backpropagation ANN technique to detect botnet attacks using the following steps, namely, dataset selection, feature extraction, data normalization, training, validation, and testing, as shown in Fig. 5.

The CTU-13 dataset from the Botnet Capture Facility Project is used in the first step. The CTU-13 contains 13 different botnet scenarios in which normal and Botnet traffic is clearly identified. The second step identifies input layer features, as shown in Table 3. Following feature selection, data values are normalized to between 0 (normal traffic) and 1 (botnet traffic).

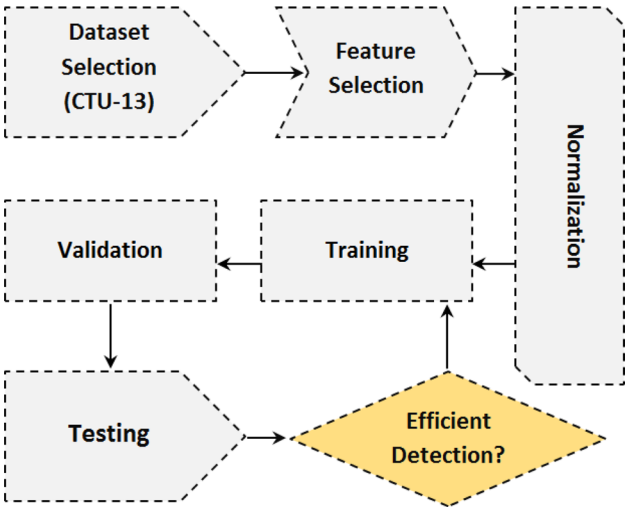


**Fig. 5** Implementation of DNN model and feed-forward backpropagation ANN technique

**Define:** α Step_Size
**Define:** β_1 ∈ [0, 1]
**Define:** β_2 ∈ [0, 1]
**Define:** $f(x)$ Objective Function (Maximization of Detection Accuracy)
**Define:** $\emptyset_O$: Initial parameter vector $O$
$m_O = 0$ $(Initial\ first\ Movement\ vector)$
$v_O = 0$ $(Initial\ second\ Movement\ vector)$
$t = 0$ $(Initial\ timestep)$
$t = t + 1$
**While** $f_x$ $Not\ converted$ ***do***
$G_t = \nabla_{\emptyset} f_t(x_{t-1})$ $(Obtain\ the\ gradients\ of\ Objective\ Function\ at\ time\ t)$
$m_t = \beta_1 . m_{t-1} + (1 - \beta_1). G_t$ $(Update\ first\ biase\ for\ first\ movement\ estimation)$
$u_t = \beta_2 . u_{t-1} + (1 - \beta_2). G_t^2$ $(Update\ first\ biase\ for\ second\ movement\ estimation)$
$\hat{m}_t = {m_t}/{(1 - \beta_1^t)}$ $(Calculate\ the\ first\ biase\ with\ respect\ to\ first\ moment\ estimation)$
$\hat{u}_t = {u_t}/{(1 - \beta_2^t)}$ $(Calculate\ the\ first\ biase\ with\ respect\ to\ second\ moment\ estimation)$
$x_t = {\hat{m}_t}/{(\sqrt{\hat{u}_t} - \epsilon)}$ $(Update\ parameters)$
**end while**
**return** $x_t$ (Output Parameters)

**Fig. 4** Optimization algorithm for the DNN-botnet detection model

**Table 3** Input layer features

| Input layer | Data attribute |
|---|---|
| X1 | Total bytes |
| X2 | Total packets |
| X3 | Duration |
| X4 | Source IP address |
| X5 | Destination IP address |
| X6 | Average bytes |
| X7 | Average packets |
| X8 | Source port |
| X9 | Destination port |

**Table 4** Flow distribution

| Flows | Purpose |
|---|---|
| 3000 | Training |
| 3500 | Validation |
| 3500 | Testing |

**Table 5** NN designs

| ANN design | No. of hidden neurons |
|---|---|
| 1 | 6 |
| 2 | 8 |
| 3 | 10 |

**Table 6** Flow distribution for developed deep-learning model

| Flows | Purpose |
|---|---|
| 8000 | Training |
| 2000 | Testing |

Following data normalization, the dataset undergoes training, validation, and testing in MATLAB 2016 version 9 using 10,000 randomly selected flows.

### 4.1 Feed-forward backpropagation ANN implementation

Table 4 shows the flow distributions for the experiment of feed-forward backpropagation algorithm. The flow distribution pattern is used for three different NN designs, as shown in Table 5. Each NN design has a similar number of inputs and flows.

### 4.2 Deep-learning neural network implementation

This subsection provides the experiment setup to implement the developed deep-learning model. Table 6 shows the flow distributions for the developed model using Tenserflow framework.

```
x_trainRaw = list(csv.reader(open("x_train.csv")))
y_trainRaw = list(csv.reader(open("y_train.csv")))
x_testRaw = list(csv.reader(open("x_test.csv")))
y_testRaw = list(csv.reader(open("y_test.csv")))

x_train = [None] * len(x_trainRaw)
y_train = [None] * len(x_trainRaw)
x_test = [None] * len(x_testRaw)
y_test = [None] * len(x_testRaw)

for i in range(0,len(x_trainRaw)):
    x_train[i] = [float(j) for j in x_trainRaw[i]]
for i in range(0,len(y_trainRaw)):
    y_train[i] = [float(j) for j in y_trainRaw[i]]
for i in range(0,len(x_testRaw)):
    x_test[i] = [float(j) for j in x_testRaw[i]]
for i in range(0,len(y_testRaw)):
    y_test[i] = [float(j) for j in y_testRaw[i]]

x_trainDF = pd.DataFrame.from_records(x_train)
y_trainDF = pd.DataFrame.from_records(y_train)
x_testDF = pd.DataFrame.from_records(x_test)
y_testDF = pd.DataFrame.from_records(y_test)

print(y_trainDF)
```

**Fig. 6** Data loading and pre-processing algorithm

Figure 6 shows the data loading and pre-processing steps before loading to the DL-Botnet Model. The first step is normalizing the entire data of all input variables such that our proposed model can smoothly integrate the data into the NN model.

Figure 7 demonstrates the algorithm's steps of the developed model trained with 8000 records from botnet data, which consist of mixed traffic of attack/non-attack. The model is optimized using Adam's optimizer as discussed earlier. The model for each run is trained for 300 iterations against the objective function, which is maximizing accuracy.

## 5 Results and discussion

This section covers the study results. In case of the feed-forward backpropagation technique, the performance of each NN design during training, validation, and testing is shown in Fig. 8. NN Design 3 (10 hidden neurons) shows the greatest accuracy. All NN designs reflect a decrease in their mean square error over time, but this decrease is likely to be reversed when the validation dataset begins to overfit the training data as it identifies random noise instead of underlying relationships.

In case of the deep-learning model, Fig. 9, demonstrates the classification accuracy achieved using our developed

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(2, activation=tf.nn.softmax))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

modelRecords = model.fit(x_trainDF.values, y_trainDF.values, epochs=300) #model records holds network data
```

**Fig. 7** Developed DNN-Botnet detection model using keras-based tensor flow framework

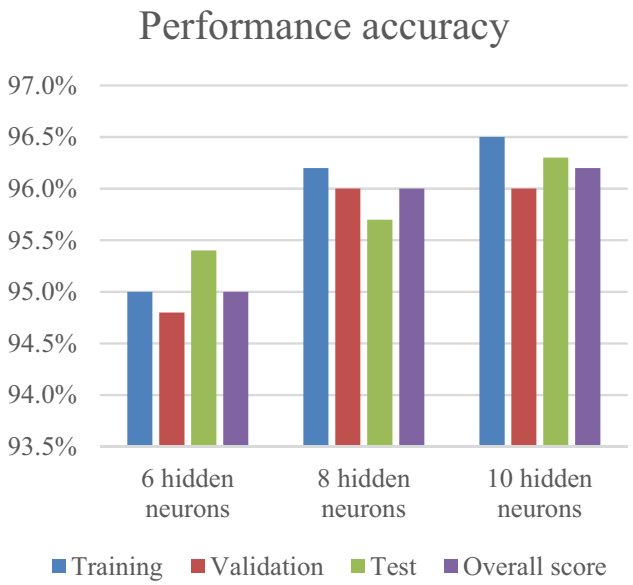


**Fig. 8** Performance of different NN designs

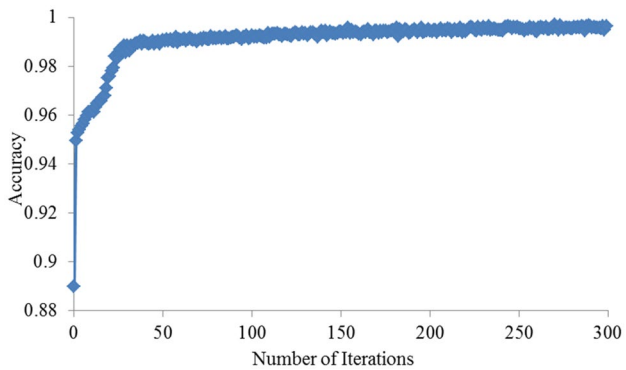


**Fig. 9** Accuracy of our proposed model over 300 iterations

DNN based Botnet detection model. It is clear that our model could converge rapidly after initial 20 iterations to achieve over 90% accuracy, until it sustained with 99.6% accuracy after 300 iterations. Hence, out of several training and testing iterations, the model was saved to be used for testing process.

Figure 10 shows the testing steps of the obtained model, out of the training process, with the remaining 2000 records. The model can achieve 99.25% accuracy on average. The model is supplied with all input variables of nine features (listed in Table 2) without providing the Y-values (class-label) that identify whether the given input is a botnet attack. Our developed model can obtain 99.25% accuracy in classifying 2000 different data traffic to its own original class (attack/non-attack). The total loss of our proposed model is 0.054 from all given Y-values, which is considered a good improvement that helps effectively in detecting Botnet attacks, compared with the state of the art. Finally, we run an experiment using our testing data to show the ability of our proposed model in predicting a botnet attack. Figure 11 shows that the model can correctly predict the class of given data to be recognized as a botnet attack (class label "1").

This paper's findings of backpropagation and deep-learning model are compared with other studies in the literature on the use of machine learning techniques to identify botnet attacks, such as SVM, decision tree, and NB. Figure 12 shows that SVM achieves 99.5% accuracy, decision tree achieves 95.2%, NB achieves 98.5%, and backpropagation achieves 96.1%. Compared with previous techniques, the DNN Model proposed in this paper achieves 99.6% for training and testing and 99.2% prediction accuracy of 2000 records. The DNN Model achieves the highest accuracy among the other approaches included in the comparison.

## 6 Conclusion

The deep learning ANN technique proposed in this paper effectively identifies botnet attacks and can be used to improve NN accuracy through hidden layer manipulation. The use of a reliable dataset is crucial to the high performance of the proposed model. This paper demonstrates that the use of deep learning in botnet detection achieves accuracies of over 99.6%, which has the highest accuracy compared with SVM, NB, or backpropagation algorithms.

```
model.save('botnet.model')

val_loss, val_acc = model.evaluate(x_testDF.values, y_testDF.values)

print(val_loss, val_acc)
```

```
2000/2000 [==============================] - 0s 109us/sample - loss: 0.0549 - acc: 0.9925
0.05488605511251353 0.9925
```

**Fig. 10** Model testing

**Fig. 11** Botnet prediction model

```
predictions = new_model.predict(x_testDF.values)

print(np.argmax(predictions[8]))
```

```
1
```

| | |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 1 |
| 9 | 1 |
| 10 | 0 |
| 11 | 0 |
| 12 | 0 |
| 13 | 0 |
| 14 | 0 |
| 15 | 0 |
| 16 | 0 |
| 17 | 0 |
| 18 | 1 |
| 19 | 0 |
| 20 | 1 |

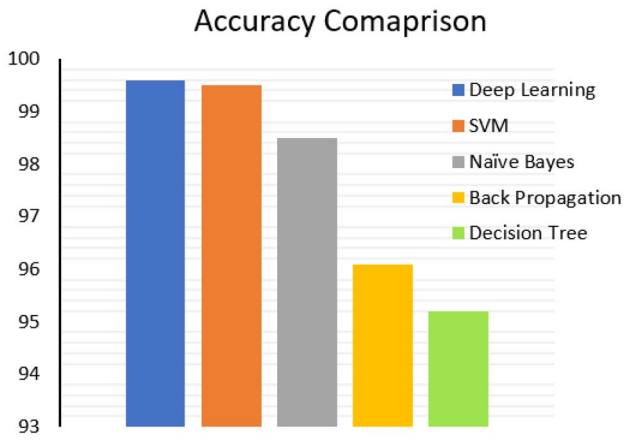


**Fig. 12** Comparison of findings

This paper recommends that other researchers examine the efficiency of the proposed model in detecting botnet attacks with different datasets. The authors plan to apply a deep learning model for detecting other malicious network threats such as DDoS attacks in a future study.

**Acknowledgements** Funding support was provided by the fund of COMSTECH-TWAS, Joint Research Grants Program for Young Scientists in OIC countries No. 14-340 RG/ITC/AS_C.

## References

Ahmed AA (2015) Investigation model for DDoS attack detection in real-time. Int J Softw Eng Comput Syst 1(1):93–105

Ahmed AA, Jantan A, Rasmi M (2013a) Service violation monitoring model for detecting and tracing bandwidth abuse. J Netw Syst Manag 21(2):218–237

Ahmed AA, Jantan A, Wan T-C (2013b) Real-time detection of intrusive traffic in QoS network domains. IEEE Secur Priv 11(6):45–53

Ahmed AA, Jantan A, Wan T-C (2016) Filtration model for the detection of malicious traffic in large-scale networks. Comput Commun 82:59–70

Al-Duwairi B, Al-Ebbini L (2010) BotDigger: a fuzzy inference system for botnet detection. In: 2010 Fifth international conference on internet monitoring and protection. pp 16–21

Al Shorman A, Faris H, Aljarah I (2019) Unsupervised intelligent system based on one class support vector machine and Grey Wolf optimization for IoT botnet detection. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-019-01387-y

Cui Z et al (2018) Detection of malicious code variants based on deep learning. IEEE Trans Ind Inform 14(7):3187–3196
Dai Q-Y, Zhang C, Wu H (2016) Research of decision tree classification algorithm in data mining. Int J Database Theory Appl 9(5):1–8
Dhaya MA, Ravi R (2020) Multi feature behavior approximation model based efficient botnet detection to mitigate financial frauds. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-020-01677-w
Garcia S, Grill M, Stiborek J, Zunino A (2014) An empirical comparison of botnet detection methods. Comput Secur 45:100–123
Gu G, Zhang J, Lee W (2008) BotSniffer: detecting botnet command and control channels in network traffic. In: Proceedings of the 15 annual network and distributed system security symposium (NDSS'08)
Jantan A, Ahmed AA (2014a) Honeybee protection system for detecting and preventing network attacks. J Theor Appl Inf Technol 64(1):38–47
Jantan A, Ahmed AA (2014b) Honey bee intelligent model for network zero day attack detection. Int J Digit Content Technol Appl 8(6):45–52
Kalaivani P, Vijaya M (2016) Mining based detection of botnet traffic in network flow. Int J Comput Sci Inf Technol Secur 6:535–540
Karasaridis A, Rexroad B, Hoeflin DA et al (2007) Wide-scale botnet detection and characterization. In: Proceedings of the first conference on first workshop on hot topics in understanding botnets (HotBots'07). pp 1–8
Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
Kolosnjaji B, Zarras A, Webster G, Eckert C (2016) Deep learning for classification of malware system call sequences. Australasian joint conference on artificial intelligence. Springer, Cham, pp 137–149
Maimó LF, Celdrán AH, Pérez MG, Clemente FJG, Pérez GM (2019) Dynamic management of a deep learning-based anomaly detection system for 5G networks. J Ambient Intell Humaniz Comput 10(8):3083–3097
Masud MM et al (2008) Cloud-based malware detection for evolving data streams. ACM Trans Manag Inf Syst (TMIS) 2(3):1–27
Moodi M, Ghazvini M (2019) A new method for assigning appropriate labels to create a 28 Standard Android Botnet Dataset (28-SABD). J Ambient Intell Humaniz Comput 10(11):4579–4593
Narang P, Ray S, Hota C, Venkatakrishnan V (2014) Peershark: detecting peer-to-peer botnets by tracking conversations. In: 2014 IEEE security and privacy workshops. pp 108–115
Rumelhart DE, Durbin R, Golden R, Chauvin Y (1995) Backpropagation: the basic theory. In: Chauvin Y, Rumelhart DE (eds) Backpropagation: theory, architectures and applications. Lawrence Erlbaum Associates, Hillsdale, New Jersey; Hove, UK, pp 1–34
Saxe J, Berlin K (2015) Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International conference on malicious and unwanted software (MALWARE). pp 11–20
Shah S, Jani H, Shetty S, Bhowmick K (2013) Virus detection using artificial neural networks. Int J Comput Appl 84(5):17–23
Singh K, Guntuku SC, Thakur A, Hota C (2014) Big data analytics framework for peer-to-peer botnet detection using random forests. Inf Sci 278:488–497
Svozil D, Kvasnicka V, Pospichal J (1997) Introduction to multilayer feed-forward neural networks. Chemom Intell Lab Syst 39(1):43–62
Vinayakumar R, Soman KP, Poornachandran P (2017) Deep android malware detection and classification. In: 2017 International conference on advances in computing, communications and informatics (ICACCI). pp 1677–1683
Wang X, Guo N, Gao F, Feng J (2019) Distributed denial of service attack defence simulation based on honeynet technology. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-019-01396-x
Ye Y, Chen L, Hou S, Hardy W, Li X (2018) DeepAM: a heterogeneous deep learning framework for intelligent malware detection. Knowl Inf Syst 54(2):265–285