Automatic Intrusion Detection System Using Deep Re-Enforcement Learning With Q-network Algorithm (DQN)

MSc Research Project
Cyber Security

Ugochukwu Nwokedi
Student ID: X21235287

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. UGOCHUKWU NWOKEDI…………………………………………………………………………………………………
Student ID:	……………X21235287……………………………………………………………………………………………..
Programme:	…CYBERSECURITY……………………………………………… **Year:** ……2022/23………………..
Module:	……ACADEMIC INTERNSHIP…………………………………………………………………………………………
Supervisor:	……MICHAEL PRIOR………………………………………………………………………………………….
Submission Due Date:	……14/08/2023……………………………………………………………………………………………
Project Title:	……AUTOMATIC INTRUSION DETECTION SYSTEM USING DEEP REINFORCEMENT LEARNING WITH Q NETWORK ALGORITHM…………………………………………………………………………………………
Word Count:	……………………………………… **Page Count**…………………………………………….……..

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

Signature: ………UGOCHUKWU NWOKEDI………………………………………………………………………………………

Date: ……12/08/2023……………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□

You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

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

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

Configuration Manual

1 Introduction

The steps and process taken in the development of this project for an Intrusion Detection System for networks is presented in this Configuration Manual. It describes all necessary settings and software tools needed to replicate the experimental setup for the project.

2 System Specification

The system configuration used in the project are:

- Operating System: Windows 10
- Processor: Intel Core i5 10 Gen
- Hard Drive: 500GB
- RAM: 8GB

3 Software Tools

Some of the software tools used to implement this project are:

- Python
- Google Colab

3.1 Software Installation

This presents the processes taken in installing the tools used.

- Download and Installation of Python 3.11.4. The download link is https://www.python.org/downloads/

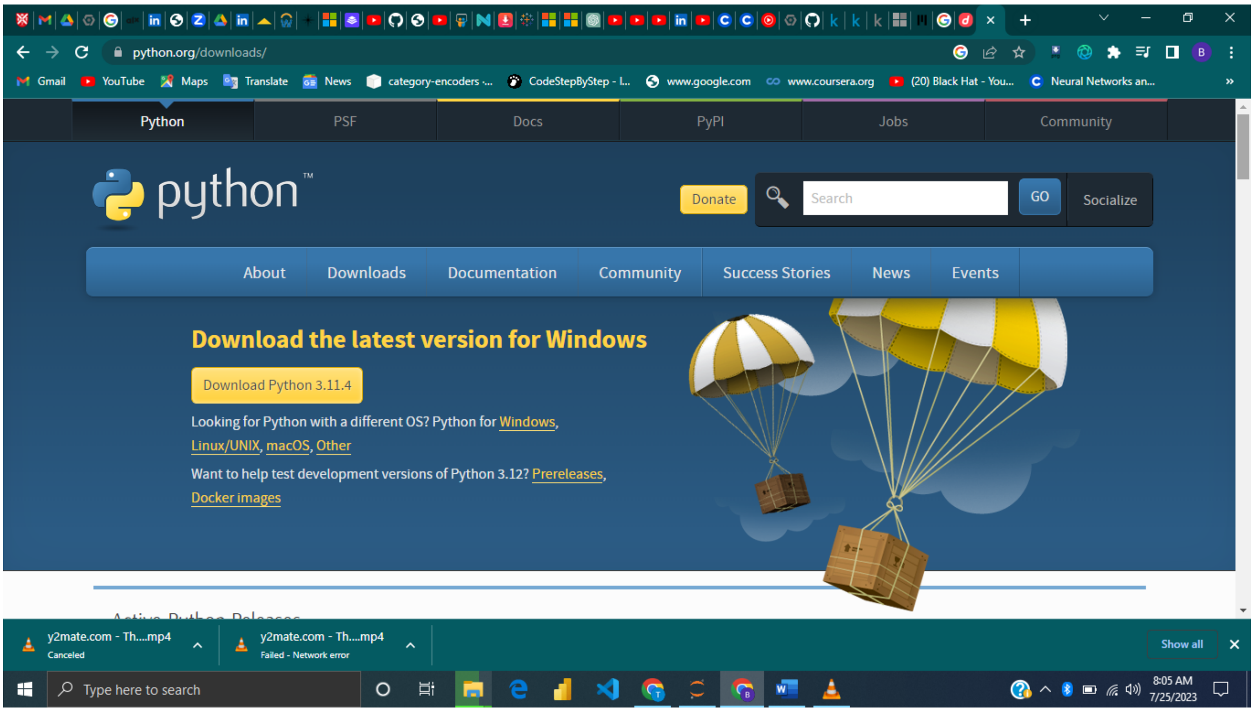

Fig 1: Python Download

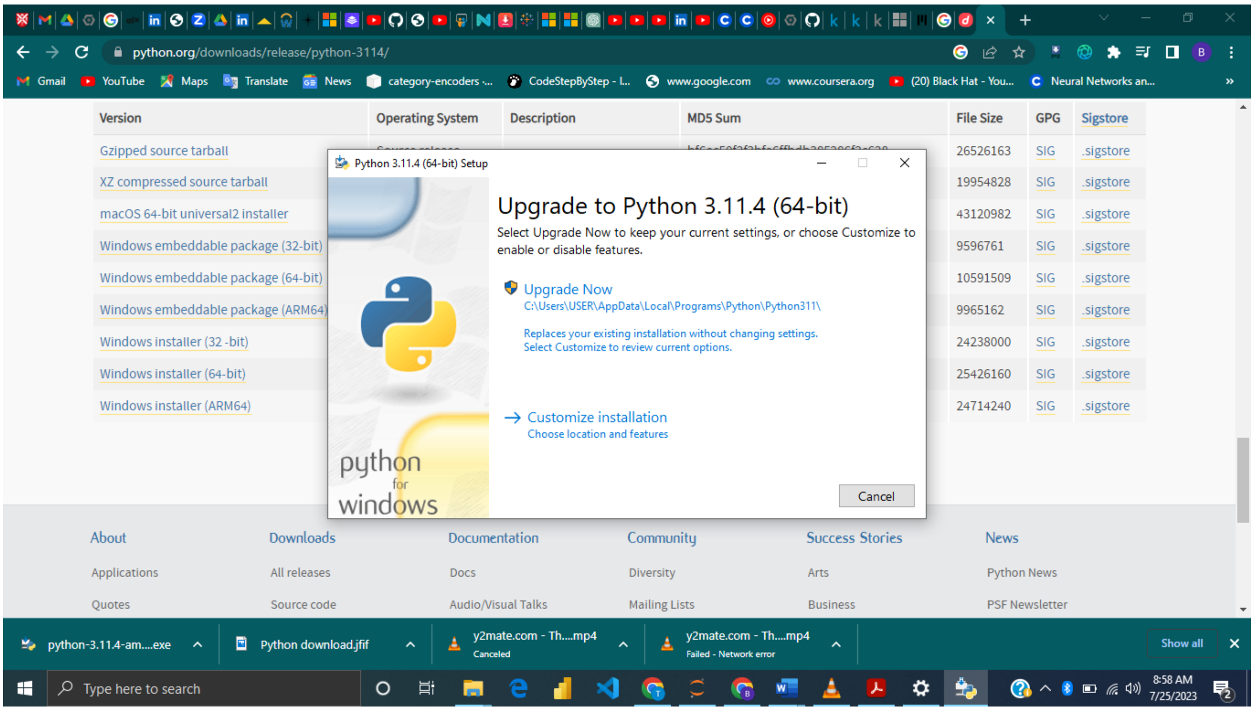

Fig 2: Python Installation

This is because I already python 3.11.3 installed on my laptop.

4 Implementation

The libraries from python used in implementing this project:

- Sckit-Learn
- Keras

- Pandas
- Numpy
- Matplotlib

```
[1] #Import dataset
    from google.colab import files
    files.upload()

    Choose Files KDD.csv
    • KDD.csv(text/csv) - 18366234 bytes, last modified: 6/4/2023 - 100% done
    Saving KDD.csv to KDD.csv
    {'KDD.csv':
    b',duration,protocol_type,service,flag,src_bytes,dst_bytes,land,wrong_fragment,urgent,hot,num_failed_logins,logged_in,num_compromised,root_shell,su_attempted,r
```

Fig 3: Mounting local file in google colab

```python
# importing required libraries
import numpy as np
import pandas as pd

import pickle # saving and loading trained model
from os import path

# importing required libraries for normalizing data
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

# importing library for plotting
import matplotlib.pyplot as plt

# importing library for support vector machine classifier
from sklearn.svm import SVC
# importing library for gradient boosting classifier
from sklearn.ensemble import GradientBoostingClassifier
# importing library for K-neares-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# importing library for Quadratic Discriminant Analysis Model
```

Fig 4: Importing libraries

1 Data preparation

In this chapter, data preparation steps are taken before passing it ro the model training and testing.
These steps includes:
Data scaling/ Normalization
One-hot encoding
Label categorization
Label encoding
Data spliting

Data Source : This is an NSL_KDD dataset

```
[3] # importing dataset
    data = pd.read_csv('KDD.csv')
```

```
[4] # print dataset
    data
```

Fig 5: Data Preparation

2.1 Data cleaning

This chapters contain the data cleaning code. We go through the process of removing the the feature that has 0 value it dataset to avoid noise in the model training.

```
[ ] #To get all the columns
    data.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 125973 entries, 0 to 125972
    Data columns (total 44 columns):
     #   Column                       Non-Null Count   Dtype
    ---  ------                       --------------   -----
     0   Unnamed: 0                   125973 non-null  int64
     1   duration                     125973 non-null  int64
     2   protocol_type                125973 non-null  object
     3   service                      125973 non-null  object
     4   flag                         125973 non-null  object
     5   src_bytes                    125973 non-null  int64
     6   dst_bytes                    125973 non-null  int64
     7   land                         125973 non-null  int64
     8   wrong_fragment               125973 non-null  int64
     9   urgent                       125973 non-null  int64
     10  hot                          125973 non-null  int64
     11  num_failed_logins            125973 non-null  int64
     12  logged_in                    125973 non-null  int64
     13  num_compromised              125973 non-null  int64
     14  root_shell                   125973 non-null  int64
     15  su_attempted                 125973 non-null  int64
     16  num_root                     125973 non-null  int64
     17  num_file_creations           125973 non-null  int64
```

Fig 6: Data preparation

1.1 Data Scaling

This cells contain the code for scaling the data into adequate size for ML algorithm.

```
[11] # selecting numeric attributes columns from data
     numeric_col = data.select_dtypes(include='number').columns
```

```
[12] # using standard scaler for normalizing
     std_scaler = StandardScaler()
     def normalization(df,col):
       for i in col:
         arr = df[i]
         arr = np.array(arr)
         df[i] = std_scaler.fit_transform(arr.reshape(len(arr),1))
       return df
```

Fig 7: Data Scaling

```
[14] # calling the normalization() function
     data = normalization(data.copy(),numeric_col)
```

```
# data after normalization
data.head()
```

	Unnamed: 0	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
0	-1.732037	-0.110249	tcp	ftp_data	SF	-0.007679	-0.004919	-0.014089	-0.089486	-0.007736
1	-1.732010	-0.110249	udp	other	SF	-0.007737	-0.004919	-0.014089	-0.089486	-0.007736
2	-1.731982	-0.110249	tcp	private	S0	-0.007762	-0.004919	-0.014089	-0.089486	-0.007736
3	-1.731955	-0.110249	tcp	http	SF	-0.007723	-0.002891	-0.014089	-0.089486	-0.007736
4	-1.731927	-0.110249	tcp	http	SF	-0.007728	-0.004814	-0.014089	-0.089486	-0.007736

5 rows × 42 columns

Fig 8: Data Scaling

2 One Hot Encoding

This cells contains code for converting our categorical feature into numeric to be easily be used for model training.

```
[16] # selecting categorical data attributes
     cat_col = ['protocol_type','service','flag']
```

```
# creating a dataframe with only categorical attributes
categorical = data[cat_col]
categorical.head()
```

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

Fig 9: One Hot Encoding

```
# one-hot-encoding categorical attributes using pandas.get_dummies() function
categorical = pd.get_dummies(categorical,columns=cat_col)
categorical.head()
```

	protocol_type_icmp	protocol_type_tcp	protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol
0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	1	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0

5 rows × 84 columns

Fig 10: One Hot Encoding

3 Label Categorization

This cells code categorize the label attacks into classes (Normal, Dos, R2L, U2R, Probe) and for easy visualization of the attack label.

```
[23] # changing attack labels to their respective attack class
     def change_label(df):
       df.label.replace(['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm'],'Dos',inplace=True)
       df.label.replace(['ftp_write','guess_passwd','httptunnel','imap','multihop','named','phf','sendmail',
            'snmpgetattack','snmpguess','spy','warezclient','warezmaster','xlock','xsnoop'],'R2L',inplace=True)
       df.label.replace(['ipsweep','mscan','nmap','portsweep','saint','satan'],'Probe',inplace=True)
       df.label.replace(['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm'],'U2R',inplace=True)

     # calling change_label() function
     change_label(data)

[25] data.label.value_counts()

     normal    67343
     Dos       45927
     Probe     11656
     R2L         995
     U2R          52
     Name: label, dtype: int64
```

Fig 11: Label Categorization

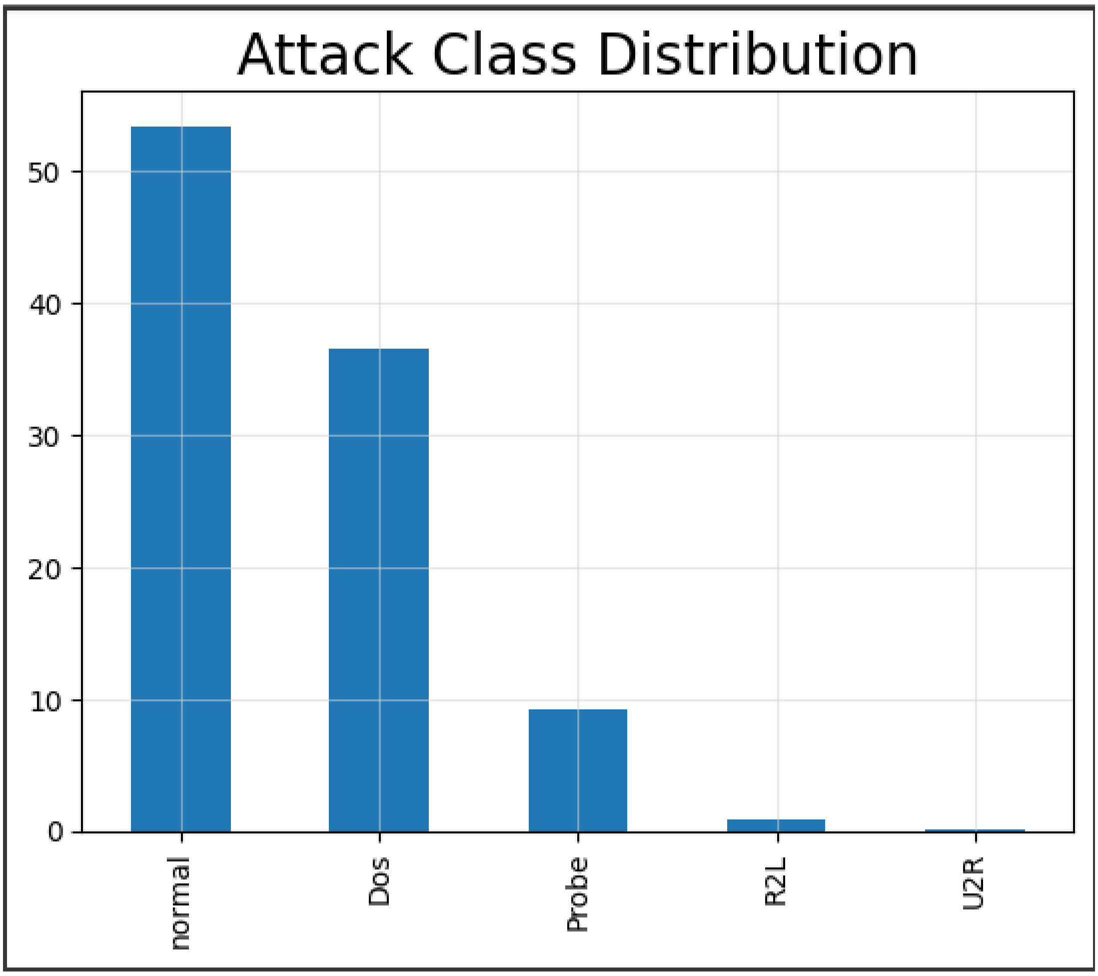

Fig 12: Attack visualization

4 Label Encoding

Label encoding is done on the label column that has categorical values to turn them into numerical values by replacing the data categories by integers starting with 0.

```
# label encoding (0,1,2,3,4) multi-class labels (Dos,normal,Probe,R2L,U2R)
le2 = preprocessing.LabelEncoder()
enc_label = multi_label.apply(le2.fit_transform)
multi_data['intrusion'] = enc_label
```

```
[31] np.save("le2_classes.npy",le2.classes_,allow_pickle=True)
```

```
[32] # one-hot-encoding attack label
     multi_data = pd.get_dummies(multi_data,columns=['label'],prefix="",prefix_sep="")
     multi_data['label'] = multi_label
     multi_data.head()
```

Fig 13: Label Encoding

5 Data Splitting

Final step to data preparation is splitting the data into training and testing sets. For this there already exists sklearn function that does all the splitting for us. This step is important so we can have data for evaluating our model and both trainand test samples should contain similar data variance.

```
[ ] X = multi_data.iloc[:,0:93].to_numpy() # dataset excluding target attribute (encoded, one-hot-encoded,original)
    Y = multi_data['intrusion'] # target attribute
```

```
[ ] # splitting the dataset 80% for training and 20% testing
    X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=42)
```

Fig 14: Data splitting

6 Model Training

6.1 Support Vector Machine

```
qsvm=SVC(kernel='poly',gamma='auto') # using kernal as polynomial for quadratic svm
qsvm.fit(X_train,y_train) # training model on training dataset
```

```
▾ SVC
SVC(gamma='auto', kernel='poly')
```

```
y_pred=qsvm.predict(X_test) # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100 # calculating accuracy of predicted data
print("QSVM-Classifier Binary Set-Accuracy is ", ac)
```

Fig 15: SVM Model Training

6.2 K-Nearest Neighbor Classfier

```
knn=KNeighborsClassifier(n_neighbors=5) # creating model for 5 neighbors
knn.fit(X_train,y_train) # training model on training dataset
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred=knn.predict(X_test)  # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100  # calculating accuracy of predicted data
print("KNN-Classifier Multi-class Set-Accuracy is ", ac)
```

Fig 16: KNN Model Training

6.3 Gradient Boosting Machine Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

# Instantiate Gradient Boosting Regressor
gbc = GradientBoostingClassifier(n_estimators=300,
                                 learning_rate=0.05,
                                 random_state=100,
                                 max_features=5 )
# Fit to training set
gbc.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05, max_features=5, n_estimators=300,
                           random_state=100)
```

```
y_pred=gbc.predict(X_test)  # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100  # calculating accuracy of predicted data
print("GBC-Classifier Multi-class Set-Accuracy is ", ac)
```

Fig 17: GBM Model Training

6.4 Quadratic Discriminant Analysis

```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train) # training model on training dataset

/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:926: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
QuadraticDiscriminantAnalysis
QuadraticDiscriminantAnalysis()
```

```
y_pred = qda.predict(X_test)  # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100  # calculating accuracy of predicted data
print("QDA-Classifier Multi-class Set-Accuracy is ", ac)
```

Fig 18: QDA Model Training

6.5 Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
import itertools

#Sckit-learn
rfc = RandomForestClassifier()

# create the RFE model and select 10 attributes
rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_train, y_train)
```

```
y_pred = rfe.predict(X_test)  # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100  # calculating accuracy of predicted data
print("QDA-Classifier Multi-class Set-Accuracy is ", ac)
```

Fig 19: Random Forest Model Training

6.6 Multi Layer Perceptron

Model definition

```
mlp = Sequential() # creating model

# adding input layer and first layer with 50 neurons
# ReLu = Rectify linear  Unit
# Unit =
mlp.add(Dense(units=50, input_dim=X_train.shape[1], activation='relu'))
# output layer with sigmoid activation
mlp.add(Dense(units=5,activation='softmax'))
```

Fig 20: MLP

Optimization and metrics

```
# defining loss function, optimizer, metrics and then compiling model
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# predicting target attribute on testing dataset
test_results = mlp.evaluate(X_test, y_test, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]*100}')
```

Fig 21: MLP

Results and Evaluation

Algorithm	Accuracy	Precision	Recall	F1 Score
SVM	93.00	92.00	93.00	92.00
KNN	98.00	98.00	98.00	98.00
GBM	97.00	97.00	97.00	97.00
QDA	47.00	73.00	47.00	53.00
Random Forest	97.45	97.00	97.00	97.00

MLP	**96.83**	**97.02**	**96.61**	**96.81**

7. Reinforcement Learning

Reinforcement learning offers a promising approach to enhance Intrusion detection system capabilities by allowing systems to learn and adapt based on feedback from their environment.
The aim of this projects is to develop an IDS using DQN algorithm to detect and classify network intrusion accurately. The IDS should be capable of learning from historical network data and making informed decisions on whether network traffic represents normal or malicious behaviors (Probe, DOS, R2L, U2R).

1.1 Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.

7.1 Building the Environment

```python
import gym
from gym import spaces
import numpy as np

class IntrusionDetectionEnv(gym.Env):
    def __init__(self, dataset_path):
        self.dataset = np.load(dataset_path,allow_pickle=True)  # Load the IDS dataset
        self.observation_space = spaces.Box(low=0, high=1, shape=(99,), dtype=np.float32)
        self.action_space = spaces.Discrete(2)
        self.state = None
        self.max_steps = self.dataset.shape[0]   # use dataset length as max step
        self.current_step = 0

    def reset(self):
        self.current_step = 0
        self.state = self.dataset[self.current_step % self.max_steps].reshape((99,))  # reshape the observation to match the environment
        return self.state

    def step(self, action):
        self.current_step += 1
        done = self.current_step >= self.max_steps
        reward = self.calculate_reward(self.state) if action == 0 else 0
        self.state = self.dataset[self.current_step % self.max_steps].reshape((99,))  # reshape the observation to match the environment
        return self.state, reward, done, {}

    def calculate_reward(self, state):
        reward = np.sum(state)
        return reward
```

Fig 22: Reinforcement Learning Environment

This cell code defines the reinforcement learning environment for the network intrusion detection and defining all the necessary component of the environment.

7.2 Building the Agent

```
from keras.optimizers import Adam
from collections import deque
import random

class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95 # discount factor
        self.epsilon = 1.0  # exploration rate
        self.epsilon_decay = 0.995
        self. epsilon_min = 0.001
        self.model = self.build_model()

    def build_model(self):
        model = Sequential()
        model.add(Dense(24, input_shape=(self.state_size,), activation = 'relu'))
        model.add(Dense(24, activation='relu'))
        model.add(Dense(self.action_size, activation= 'linear'))
        model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
        return model

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        act_values = self.model.predict(state)
        return np.argmax(act_values[0])
```

```
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        target = reward
        if not done:
            target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])
        target_f = self.model.predict(state)
        target_f[0][action] = target
        self.model.fit(state, target_f, epochs=1, verbose=0)
    if self.epsilon > self.epsilon_min:
      self.epsilon *= self.epsilon_decay
```

Fig 23: Reinforcement Learning Agent

This cell codes defines the agent that will perform some certain actions and explore the environment.

7.3 Implementing the data

```
# create the environment
dataset_path = '/content/data.npy'
env = IntrusionDetectionEnv(dataset_path)

# set the state and action sizes
state_size = env.observation_space.shape[0]
action_size = env.action_space.n
```

```
# create the DQN agent
agent = DQNAgent(state_size, action_size)

# training loop
num_episodes = 10
batch_size = 320
episode_rewards = []  # track rewards per episode
```

Fig 24: data implementation

7.5 Tracking the process of the training

```
for episode in range(num_episodes):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    done = False
    total_reward = 0

    while not done:
        action =  agent.act(state)
        next_state, reward, done, _ = env.step(action)
        next_state = np.reshape(next_state, [1, state_size])
        agent.remember(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward

    episode_rewards.append(total_reward)   #track the episode reward

    print("Episode:", episode, "Total Reward", total_reward)

    if len(agent.memory) > batch_size:
        agent.replay(batch_size)
```

Fig 25: Tracking with Episodes

This code cells tracking the training process by looping through the episodes.

7.6 Plotting the Learning Curve

```
# plotting the learning curve
plt.plot(episode_rewards)
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.title('Learning Curve')
plt.show()
```

Fig 25: Learning Curve

This cell code measures the evaluation of the performance of the implemented DQN algorithm and monitoring the convergence of the agent's total rewards