NORTH CAROLINA AGRICULTURAL AND TECHNICAL STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

IDSGAN: GENERATIVE ADVERSARIAL NETWORKS FOR ATTACK GENERATION AGAINST INTRUSION DETECTION

AUTHORS: ZILONG LIN [1], YONG SHI [1,2], ZHI XUE [1,2]
[1] SCHOOL OF CYBER SECURITY, SHANGHAI JIAO TONG UNIVERSITY, SHANGHAI, P.R. CHINA
[2] SHANGHAI KEY LABORATORY OF INTEGRATED ADMINISTRATION TECHNOLOGIES FOR INFORMATION SECURITY, SHANGHAI, P.R. CHINA
PUBLISHED IN ARXIV JUNE 16,2019

CLASS INSTRUCTOR:
DR. MAHMOUD N. MAHMOUD

DANIEL BLANKSON
04/28/2020

AGGIES DO

OUTLINE

- Intrusion Detection Systems (IDS)
  - Overview
  - Types
  - Subtypes
- Generative Adversarial Networks(GAN)
  - Overview
  - Generator
  - Discriminator
  - Loss Function
- Literature Review
- IDSGAN
- Dataset
- Experiment
- Results

IDS OVERVIEW

- Unscrupulous elements constantly look to stage an intrusion through a variety of ways
  - » Ransomware
  - » insider threats and
  - » sync errors
- IDS monitors network traffic for suspicious activities and known threats.
  - » *Monitor systems*
  - » *Research system logs*
  - » *Identify the design of typical attacks*

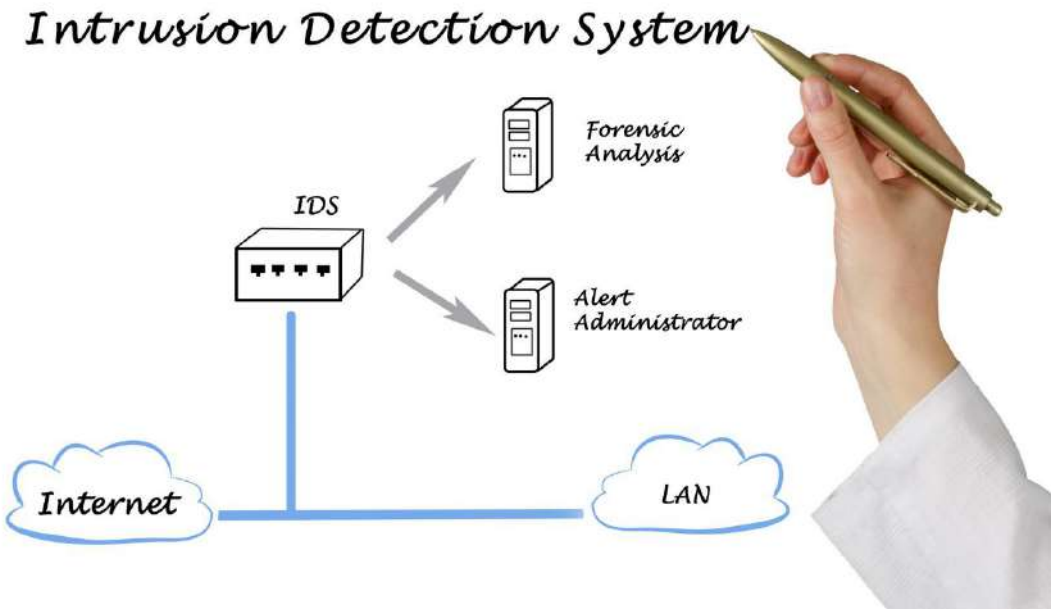


Image source: https://candid.technology/intrusion-detection-systems-benefits-types-modes-of-operation/

TYPES

- An intrusion detection system is broadly categorized based on where the IDS is placed: network or host.
- **Network Intrusion Detection System (NIDS)**
  - » Placed at crucial points in the network to inspect traffic from all devices on the network
    - For instance, on the subnet where firewalls are located to detect Denial of Service (DoS) and other such attacks

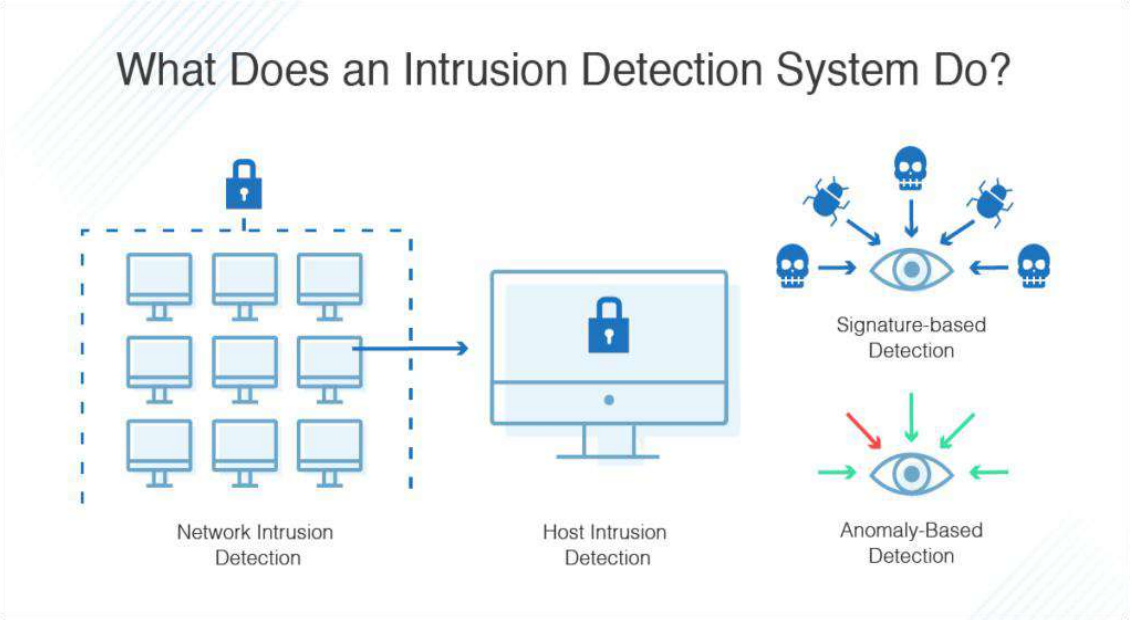


Image source: https://www.dnsstuff.com/ids-vs-ips

TYPES

- **Host Intrusion Detection System (HIDS)**
  - » monitors and analyzes system configuration and application activity for devices running on the enterprise network
  - » Installed on any device

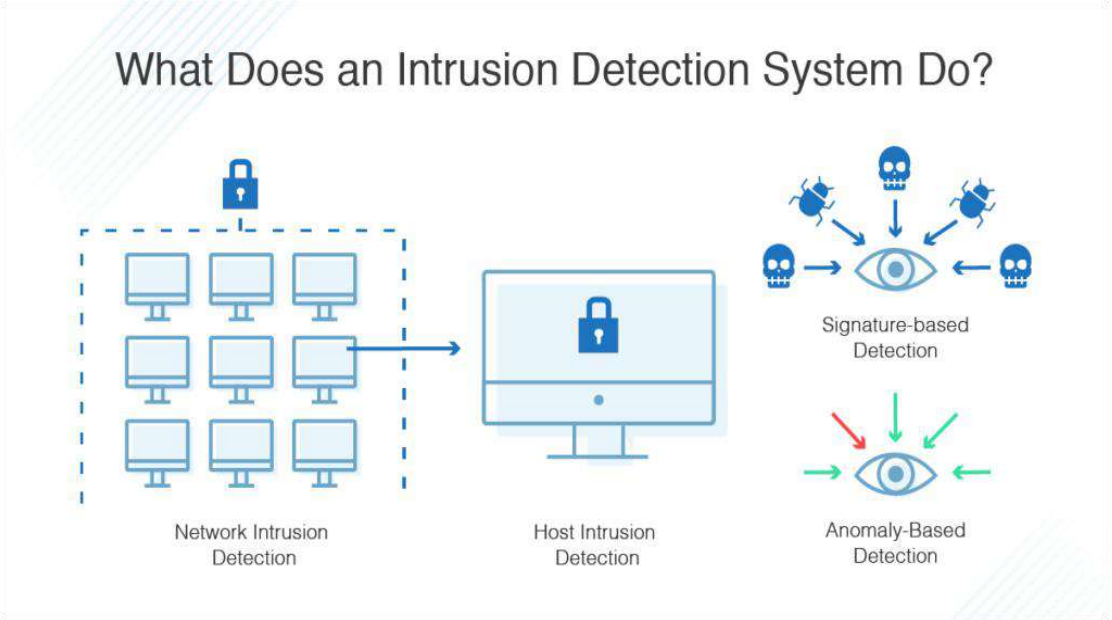


Image source: https://www.dnsstuff.com/ids-vs-ips

SUBTYPES

- Anomaly-based
  - » detect and adapt to unknown attacks
    - uses machine learning model
      - compare new behavior against this trust model
- Signature-based
  - » looks for specific patterns, such as
    - byte sequences in network traffic
    - known malicious instruction sequences used by malware

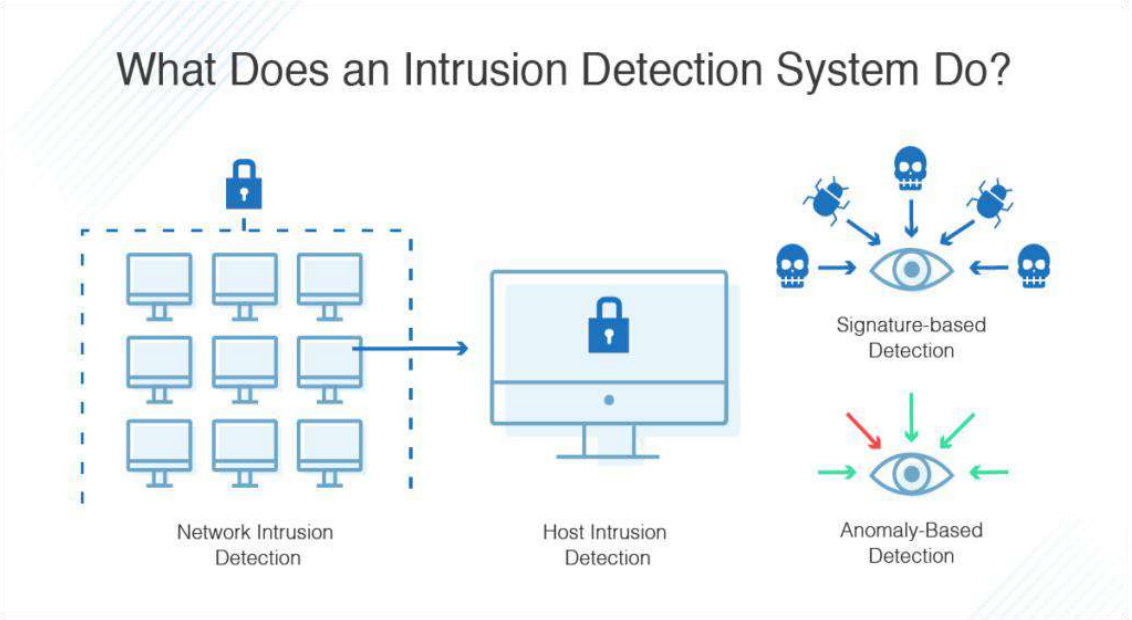


Image source: https://www.dnsstuff.com/ids-vs-ips

SUBTYPES

- NIDS works in real-time
  - » tracks live data and flags issues as they happen
- HIDS examines historical data
- Use both HIDS and NIDS as they complement each other

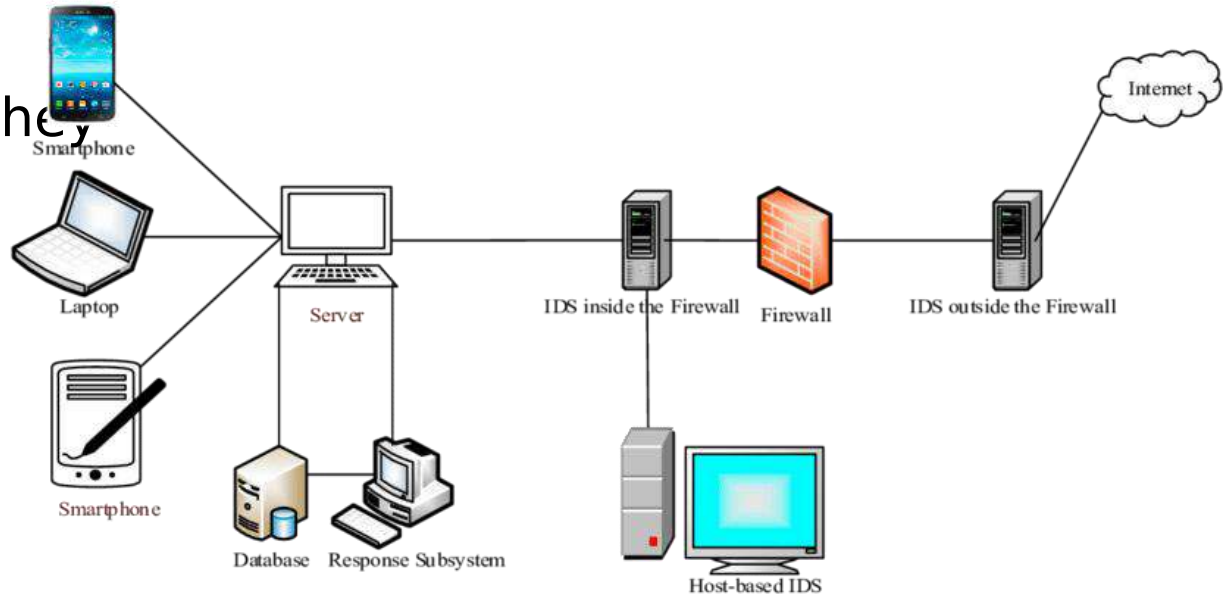


Image source: https://www.researchgate.net/figure/Intrusion-detection-system-architecture-37_fig2_315662151

GENERATIVE ADVERSARIAL NETWORKS (GAN)

Yann LeCun described it as “the most interesting idea in the last 10 years in Machine Learning”

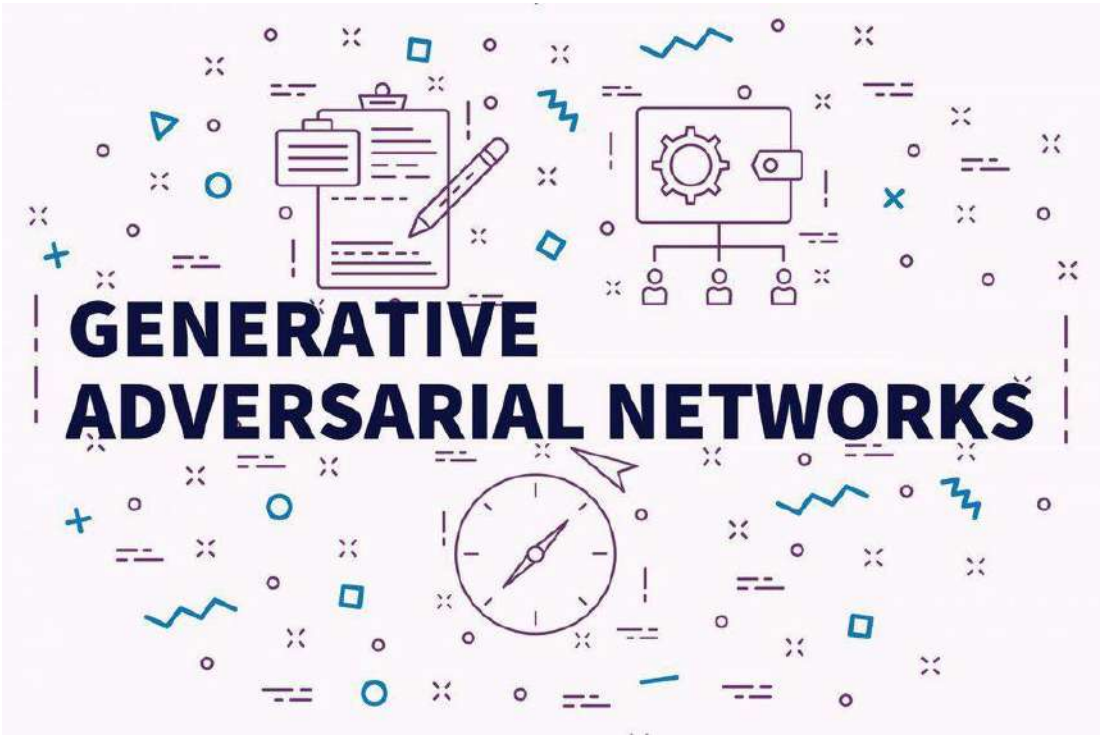


Image source: https://www.forbes.com/sites/bernardmarr/2019/06/12/artificial-intelligence-explained-what-are-generative-adversarial-networks-gans/#16abe1727e00

GENERATIVE ADVERSARIAL NETWORKS (GAN)

- Discriminative models learn the boundary between classes
- Generative models model the distribution of individual classes

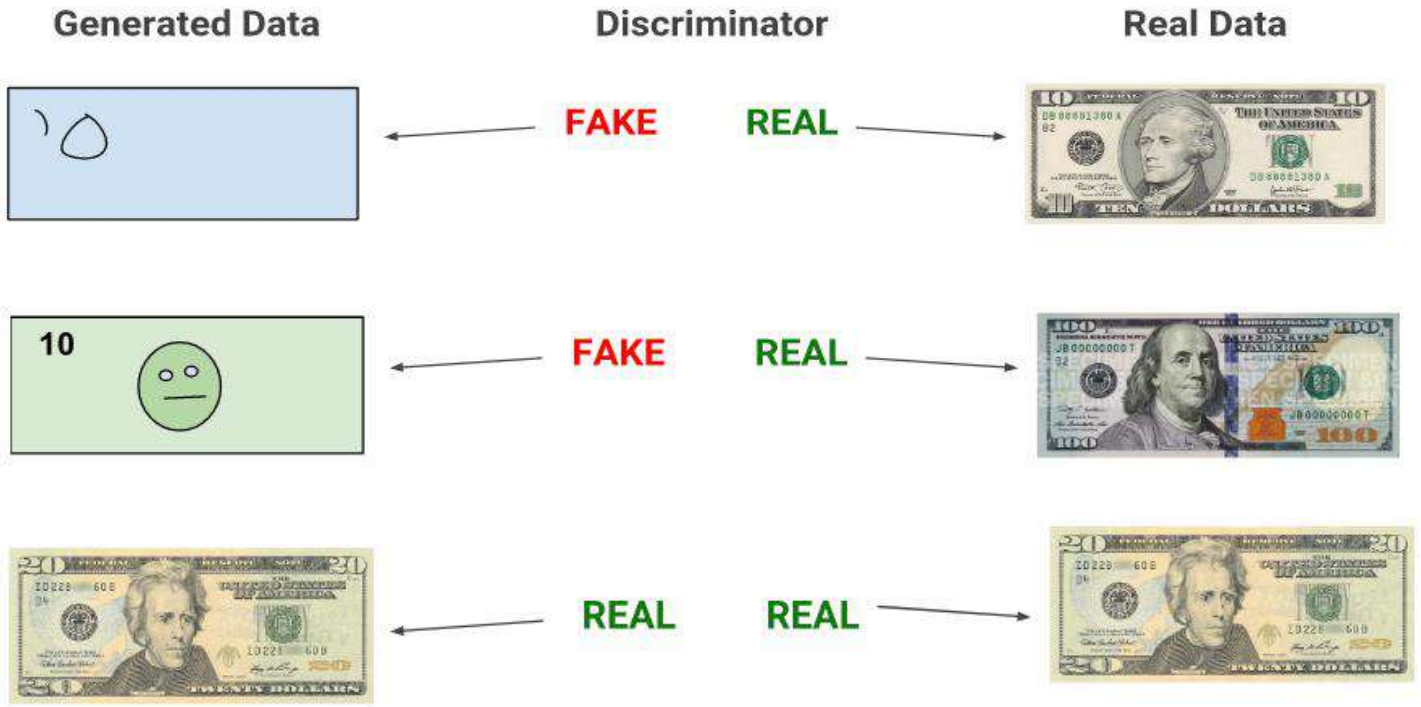


Image source: https://developers.google.com/machine-learning/gan/gan_structure

GENERATIVE ADVERSARIAL NETWORKS (GAN)

- The discriminator in a GAN is simply a classifier.
- It tries to distinguish real data from the data created by the generator.

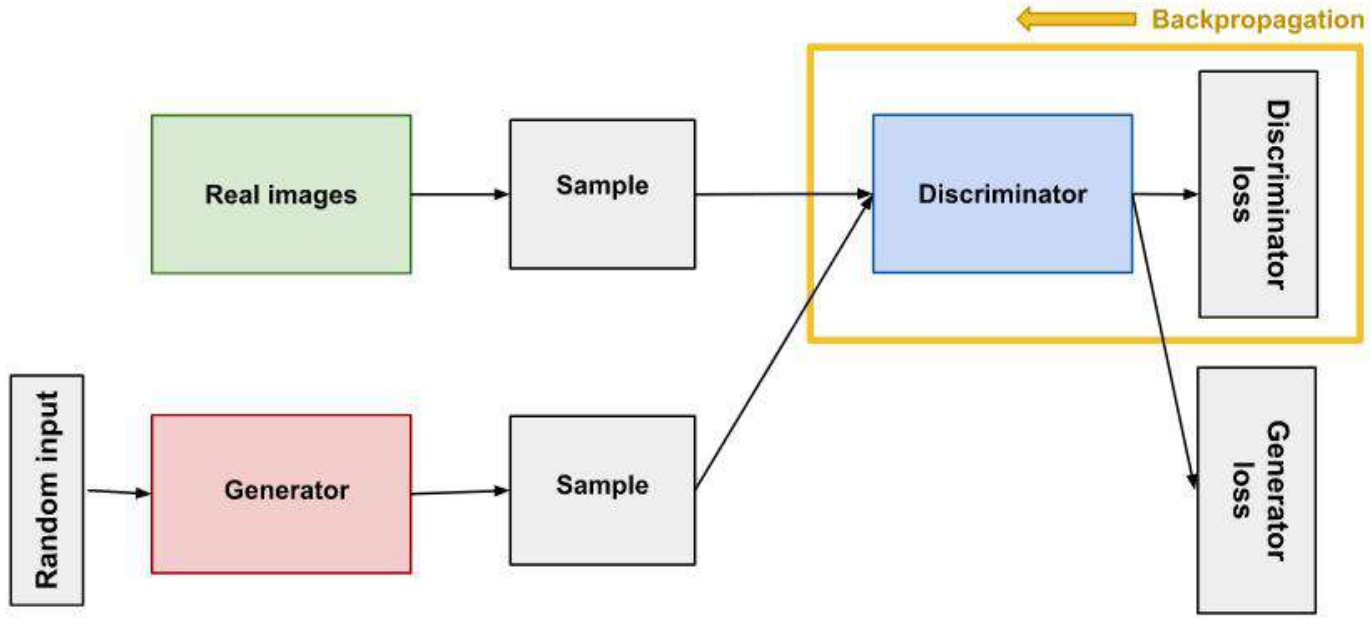
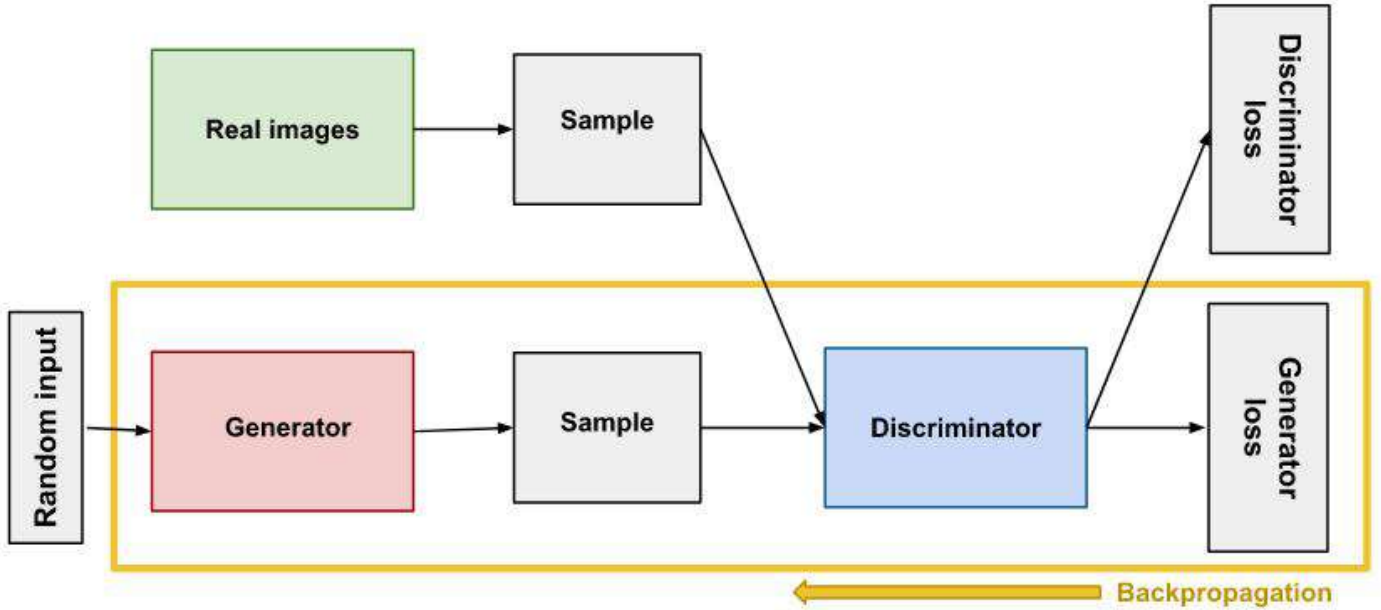


Image source: https://developers.google.com/machine-learning/gan/gan_structure

GENERATIVE ADVERSARIAL NETWORKS (GAN)

- The portion of the GAN that trains the generator includes:
  - random input
  - generator network, discriminator network,
  - discriminator output
  - generator loss,



$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

Image source: https://developers.google.com/machine-learning/gan/gan_structure

REVIEW OF LITERATURE

- Grosse et al. [1] proposed to apply the algorithm based on the forward derivative of the attacked neural networks to craft adversarial Android malware examples with the function of the malware preserved
  - The classifier achieves up to 97% accuracy with 7.6% false negatives and 2% false positives on the DREBINS Dataset

- Hu and Tan [2] proposed a GAN framework to generate adversarial malware examples for black-box attacks
  - Experimental results show that almost all of the adversarial examples generated by MalGAN successfully bypass the detection algorithms

IDSGAN OVERVIEW

- Problem: Robustness of the IDS is questionable when it faces the adversarial attacks.
  - » To improve the detection system, more potential attack approaches should be researched
- IDSGAN, is proposed to generate the adversarial attacks, which can deceive and evade the intrusion detection system.
  - » Black-box Attack

DATASET

- 43 features
  - 41 Input Features
  - 2 Output Features
    - Labels- normal or **attack**
    - Scores- the severity of the traffic input itself.
- Classes of Attacks:
  - Denial of Service (DoS),
  - Probe,
  - User to Root(U2R), and
  - Remote to Local (R2L)

Dataset	Number of Records:					
	Total	Normal	DoS	Probe	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.11%)	52 (0.04%)	995 (0.85%)
KDDTest+	22544	9711 (43%)	7458 (33%)	2421 (11%)	200 (0.9%)	2654 (12.1%)

Image source: https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657

DATASET

- DoS tries to shut down traffic flow to and from the target system
  - » flooded with an abnormal amount of traffic
  - » The system can’t handle, and shuts down to protect itself.
  - » This prevents normal traffic from visiting a network
- Probe or surveillance tries to get information from a network
- U2R starts off with a normal user account and tries to gain access to the system or network, as a super-user (root).
- R2L tries to gain local access to a remote machine

Classes:	DoS	Probe	U2R	R2L
Sub-Classes:	• apache2 • back • land • neptune • mailbomb • pod • processtable • smurf • teardrop • udpstorm • worm	• ipsweep • mscan • nmap • portsweep • saint • satan	• buffer_overflow • loadmodule • perl • ps • rootkit • sqlattack • xterm	• ftp_write • guess_passwd • httptunnel • imap • multihop • named • phf • sendmail • Snmpgetattack • spy • snmpguess • warezclient • warezmaster • xlock • xsnoop
Total:	11	6	7	15

Image source: https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657

FEATURES

Feature Categories	Description	Features
Intrinsic	Derived from the header of the packet	1–9
Content features	Hold information about the original packets	10–22
Time-based features	Hold the analysis of the traffic input over a two-second window	23–31
Host-based features	Analyzes over a series of connections made	32–41

DATASET

Dataset	Number of Records:					
	Total	Normal	DoS	Probe	U2R	R2L
KDDTrain+20%	25192	13449 (53%)	9234 (37%)	2289 (9.16%)	11 (0.04%)	209 (0.8%)
KDDTrain+	125973	67343 (53%)	45927 (37%)	11656 (9.11%)	52 (0.04%)	995 (0.85%)
KDDTest+	22544	9711 (43%)	7458 (33%)	2421 (11%)	200 (0.9%)	2654 (12.1%)

Classes:	DoS	Probe	U2R	R2L
Sub-Classes:	• apache2 • back • land • neptune • mailbomb • pod • processtable • smurf • teardrop • udpstorm • worm	• ipsweep • mscan • nmap • portsweep • saint • satan	• buffer_overflow • loadmodule • perl • ps • rootkit • sqlattack • xterm	• ftp_write • guess_passwd • httptunnel • imap • multihop • named • phf • sendmail • Snmpgetattack • spy • snmpguess • warezclient • warezmaster • xlock • xsnoop
Total:	11	6	7	15

Image source: https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657

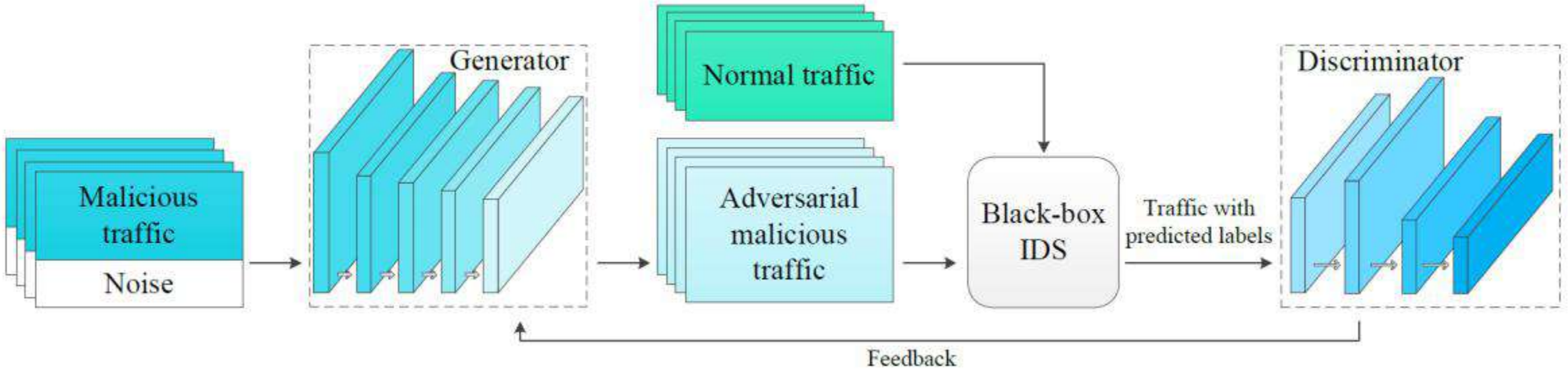
DATA PRE-PROCESSING

- discrete features-9
  - » nonnumeric values-3
    - protocol type, service and flag
      - protocol type has 3 types of attributes: TCP, UDP and ICMP, which are encoded into the numeric discrete values as 1, 2 and 3.
  - » binary values-6
- Dataset is normalized

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

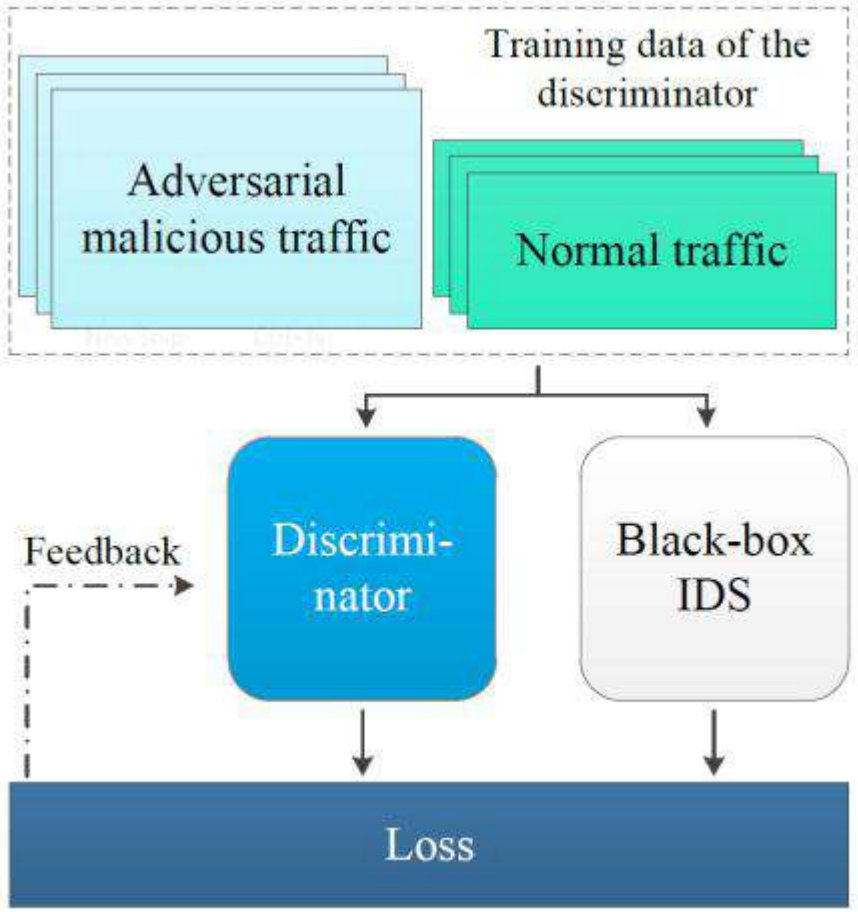
GENERATOR

- Generator plays the role of the adversarial malicious traffic example generation for the evasion attack.
  - To transform an original example into an adversarial example, each input vector of the traffic examples should consists of an m-dimensional original example vector M and an n-dimensional noise vector N.



DISCRIMINATOR

- Discriminator is used imitate the black-box IDS
  - This helps the generator training because the adversarial attacks can try to bypass the IDS during the training of the generator.
  - As a classification tool, the discriminator classifies the outputs of the generator and supplies the feedback to the generator.



EXPERIMENT

- Deep learning framework- Pytorch
  - a Linux PC with Intel Core i7-2600
  - batch size=64
  - Epochs=100
  - Learning rates of the generator and the discriminator are both 0.0001.
- algorithms of the black-box IDS
  - Support Vector Machine (SVM),
  - Naive Bayes (NB),
  - Multilayer Perceptron (MLP),
  - Logistic Regression (LR),
  - Decision Tree (DT),
  - Random Forest (RF) and
  - K-Nearest Neighbor (KNN).

- Black-box IDS training set:
  - 1/2 of the records in KDDTrain+
- Discriminator:
  - 1/2 of the normal traffic records in the other half of KDDTrain+ and the adversarial malicious traffic examples from the generator
- For the experimental metrics, the detection rate and the evasion increase rate are measured

$$DR = \frac{Num.\ of\ correctly\ detected\ Attacks}{Num.\ of\ All\ the\ attacks}$$

$$EIR = 1 - \frac{Adversarial\ detection\ rate}{Original\ detection\ rate}$$

UNMODIFIED FEATURES

- Each category of attacks has its functional features
  - » the basic function of this attack.
- To avoid invalidating the traffic, we must keep the functional features of each attack unchanged.

Attack	Functional features			
	Intrinsic	Content	Time-based traffic	Host-based traffic
Probe	✓		✓	✓
DoS	✓		✓	
U2R	✓	✓		
R2L	✓	✓		

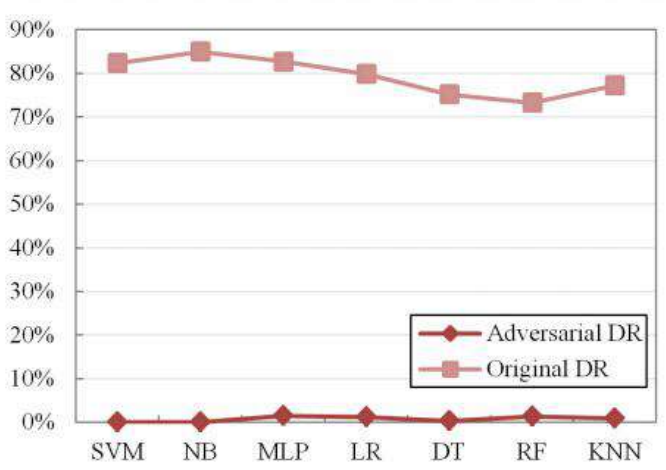
RESULTS

Adding unmodified features	Attack	Metric	SVM	NB	MLP	LR	DT	RF	KNN
—	DoS	Original DR (%)	82.37	84.94	82.70	79.85	75.13	73.28	77.22
—	U2R & R2L	Original DR (%)	0.68	6.19	4.54	0.64	12.66	2.44	5.69
×	DoS	Adversarial DR (%)	0.04	0.00	1.56	1.23	0.38	1.32	0.92
		EIR (%)	99.95	100.00	98.11	98.46	99.49	98.20	98.81
×	U2R & R2L	Adversarial DR (%)	0.00	0.71	0.00	0.00	0.03	0.00	0.00
		EIR (%)	100.00	88.53	100.00	100.00	99.76	100.00	100.00
✓	DoS	Adversarial DR (%)	25.66	0.62	48.44	34.00	10.49	25.98	70.97
		EIR (%)	68.85	99.27	41.43	57.42	86.04	64.55	8.09
✓	U2R & R2L	Adversarial DR (%)	0.01	4.96	0.92	0.00	0.65	0.00	0.27
		EIR (%)	98.51	19.87	79.74	100.00	94.87	100.00	95.25

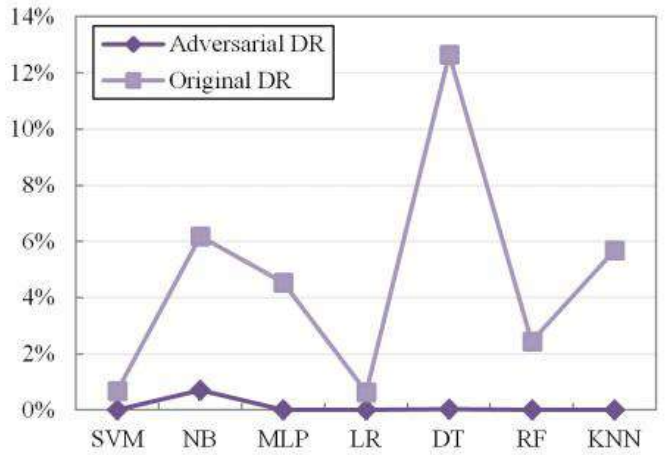
- The performance of IDSGAN under DoS, U2R and R2L. The first and second lines in the table are the black-box IDS's original detection rates to the original testing set. "×" in "adding unmodified features' means that the lines are the performance of IDSGAN with only the functional features unmodified. "X" in "adding unmodified features" means that the lines are the performance of IDSGAN with the added unmodified features.

RESULTS

- The distances between the original detection rates and the adversarial detection rates of these attacks are also large and evident.



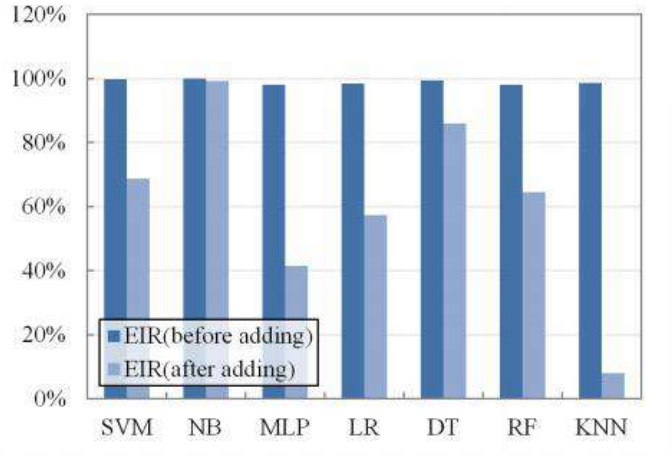
(a)



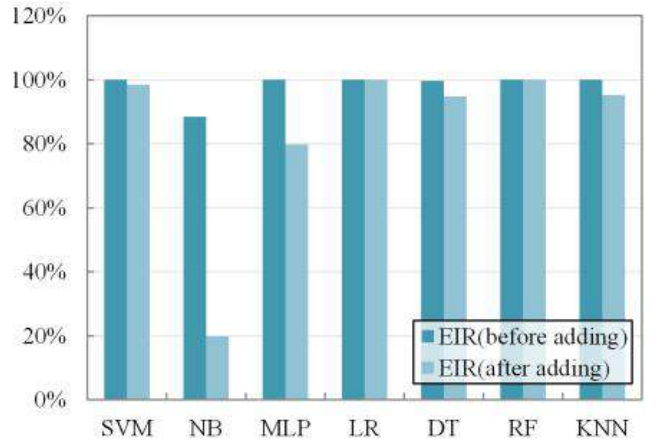
(b)

RESULTS

- Compared with the experiments with only the functional features unmodified, the evasion increase rates in contrast experiments decline or maintain.
- With the increase of the amount of unmodified features, more original information in one traffic record is retained after the adversarial generation. With the help of this increased information, the black-box IDS is able to detect based on more pristine and precise information, making more accurate judgments in the test.



(a)



(b)

CODE SNIPPET

```python
import pandas as pd
import numpy as np
#import adabound
import torch as th
from torch.autograd import Variable as V
from torch import nn,optim
from preprocessing import  preprocess2,create_batch1
from model_class import Blackbox_IDS
import matplotlib.pyplot as plt

train = pd.read_csv("KDDTrain+.csv")
test = pd.read_csv("KDDTest+.csv")
trainx,trainy,testx,testy = preprocess2(train,test)
input_dim = trainx.shape[1]
output_dim = 2
batch_size = 1024
tr_N = len(trainx)
te_N = len(testx)
ids_model = Blackbox_IDS(input_dim,output_dim)
opt = optim.Adam(ids_model.parameters(),lr=0.001)
loss_f = nn.CrossEntropyLoss()
max_epoch = 50
train_losses, test_losses = [],[]

def train(x,y):
    ids_model.train()
    batch_x, batch_y = create_batch1(x,y,batch_size)
    run_loss = 0
    for x,y in zip(batch_x,batch_y):

        ids_model.zero_grad()
        x = V(th.Tensor(x),requires_grad = True)
        y = V(th.LongTensor(y))
        out = ids_model(x)
        loss = loss_f(out,y)
        run_loss += loss.item()
        loss.backward()
        opt.step()
    return run_loss/tr_N

def test(x,y):
    ids_model.eval()
    batch_x, batch_y = create_batch1(x,y,batch_size)
    run_loss = 0

    with th.no_grad():
        for x,y in zip(batch_x,batch_y):
            x = th.Tensor(x)
            y = th.LongTensor(y)
            out = ids_model(x)
            loss = loss_f(out,y)
            run_loss += loss.item()
    return run_loss/te_N
def main():
    print("IDS start training")
    print("-"*100)
    for epoch in range(max_epoch):
        train_loss = train(trainx,trainy)
        test_loss = test(testx,testy)
        train_losses.append(train_loss)
        test_losses.append(test_loss)
        print(f"{epoch} : {train_loss} \t {test_loss}")

    print("IDS finished training")

    th.save(ids_model.state_dict(), 'model/IDS.pth')
    plt.plot(train_losses,label = "train")
    plt.plot(test_losses, label = "test")
    plt.legend()
    plt.show()
if __name__ == "__main__":
    main()
```

```python
import torch as th
from torch import nn
class Blackbox_IDS(nn.Module):
    def __init__(self,input_dim, output_dim):
        super().__init__()
        self.layer = nn.Sequential(
            #nn.BatchNorm1d(input_dim),
            nn.Linear(input_dim, input_dim*2),
            nn.Dropout(0.6),
            #nn.ELU(),
            nn.LeakyReLU(True),
           # nn.BatchNorm1d(input_dim*2),
            nn.Linear(input_dim *2, input_dim *2),
            nn.Dropout(0.5),
           # nn.ELU(),
#            nn.ReLU(True),
            nn.LeakyReLU(True),
          # nn.BatchNorm1d(input_dim*2),
            nn.Linear(input_dim *2, input_dim//2),
            nn.Dropout(0.5),
#            nn.ReLU(True),
        #   nn.ELU(),
            nn.LeakyReLU(True),
           #nn.BatchNorm1d(input_dim//2),
            nn.Linear(input_dim//2,input_dim//2),
            nn.Dropout(0.4),
        #    nn.ELU(),
#            nn.ReLU(True),
            nn.LeakyReLU(True),

             nn.Linear(input_dim//2,output_dim),
        )
        #nn.init.kaiming_normal_(self.layer.weight)
        self.output = nn.Sigmoid()
        #self.output = nn.Softmax()
    def forward(self,x):
        x = self.layer(x)
        return x
```

```python
class Generator(nn.Module):
    def __init__(self,input_dim, output_dim):
        super(Generator, self).__init__()
        self.layer = nn.Sequential(
            nn.Linear(input_dim, input_dim//2),
            nn.ReLU(True),
            nn.Linear(input_dim //2, input_dim//2),
            nn.ReLU(True),
            nn.Linear(input_dim // 2, input_dim//2),
            nn.ReLU(True),
            nn.Linear(input_dim//2,input_dim//2),
            nn.ReLU(True),
            nn.Linear(input_dim//2,output_dim),
        )
    def forward(self,x):
        x = self.layer(x)
        return th.clamp(x,0.,1.)

class Discriminator(nn.Module):
    def __init__(self,input_dim, output_dim):
        super(Discriminator, self).__init__()

        self.layer = nn.Sequential(
            nn.Linear(input_dim, input_dim*2),
            nn.LeakyReLU(True),
            nn.Linear(input_dim * 2, input_dim *2),
            nn.LeakyReLU(True),
            nn.Linear(input_dim*2 , input_dim*2),
            nn.LeakyReLU(True),
            nn.Linear(input_dim*2,input_dim//2),
            nn.LeakyReLU(True),
            nn.Linear(input_dim//2,output_dim),
        )

    def forward(self,x):
        return self.layer(x)
```

REFERENCES

1. Lin, Z., Shi, Y., & Xue, Z. (2018). IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection. ArXiv Preprint ArXiv:1809.02077.
2. Grosse, Kathrin & Papernot, Nicolas & Manoharan, Praveen & Backes, Michael & McDaniel, Patrick. (2016). Adversarial Perturbations Against Deep Neural Networks for Malware Classification.
3. Hu, Weiwei & Tan, Ying. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN.


QUESTIONS