数据集：CIC Tor-nonTor 数据集，提取流长度序列。

预先训练模型：

- dnn.model：acc = 94.77%
- cnn.model：acc = 90.02%
- lstm.model：acc = 97.68%

基于 CleverHans 库（v4.0.0 兼容 TF2），分别使用 FGSM、BIM、MIM 和 CW 进行对抗样本实验。

安装 tensorflow 库：

```
# pip install tensorflow=2.0.0a0
```

安装 cleverhans 库：

```
# pip install git+https://github.com/tensorflow/cleverhans.git#egg=cleverhans
```

从本地加载模型：

```
from tensorflow.keras.models import load_model
from tensorflow.keras import Model

model = load_model("/root/WorkPlace/model/dnn.model")
logits_model = Model(model.input, model.layers[-1].output)
```

读取负样本：

```
import numpy as np
import pandas as pd
import tensorflow as tf

X_test = np.empty(shape(0, 10))
names = ["browsing", "email", "chat", "streaming", "file", "voip", "p2p"]

for name in names:

    data = pd.read_csv("/root/WorkPlace/tor_csv/tor_" + name + "_lenseq.csv", nrows=100, index_col=0)
    _X_test = data.values
    X_test = np.vstack((X_test, _X_test)).astype("float32")
```

CNN 的输入：

```
X_test = tf.reshape(X_test, (X_test.shape[0], 10, 1))
```

LSTM 的输入：

```
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

FGSM 方法

初始化参数：

```
from cleverhans.tf2.attacks.fast_gradient_method import fast_gradient_method

fgsm_params = {"norm": np.inf, "eps": 20, "clip_min": 0.0, "clip_max": 200.0}
```

生成对抗样本：

```
adv_x = fast_gradient_method(logits_model, X_test, **fgsm_params)
adv_pred = model.predict(adv_x)
```

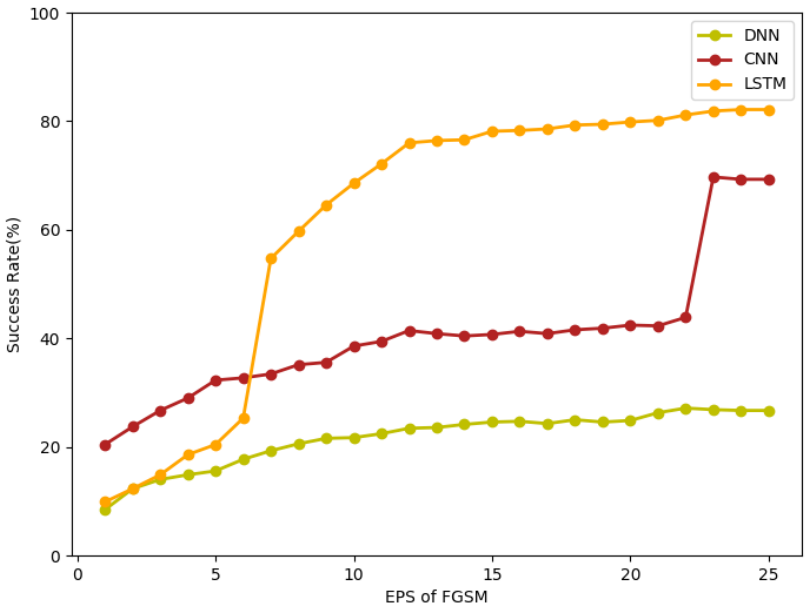
统计攻击成功率：

```
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

FGSM 攻击成功率：

- DNN：24.86%
- CNN：42.43%
- LSTM：79.86%

不同 eps 下的攻击成功率：



BIM 方法

初始化参数：

```
from cleverhans.tf2.attacks.basic_iterative_method import basic_iterative_method

bim_params = {"norm": np.inf, "eps": 20, "eps_iter": 1, "nb_iter": 1000, "clip_min": 0.0, "clip_max":
200.0, "sanity_checks": False}
```

生成对抗样本：

```
adv_x = basic_iterative_method(logits_model, X_test, **bim_params)
adv_pred = model.predict(adv_x)
```

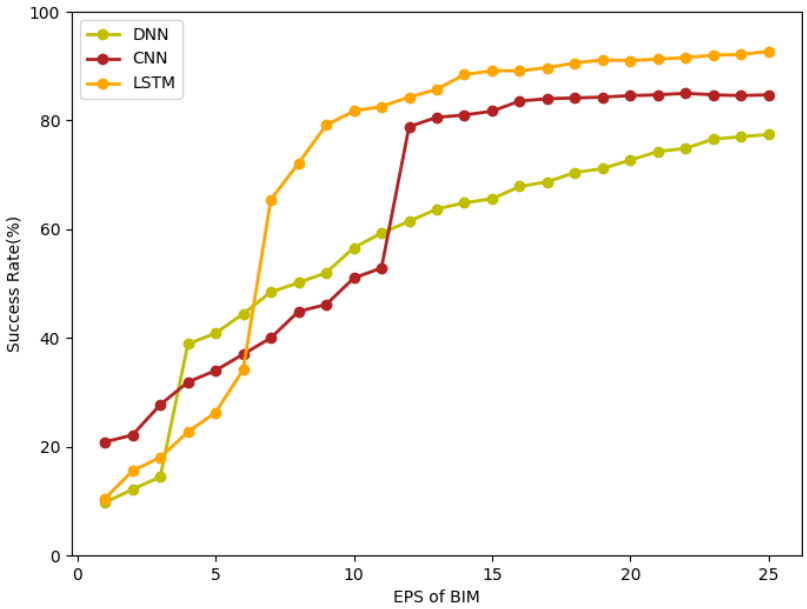
统计攻击成功率：

```
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

BIM 攻击成功率：

- DNN：72.71%
- CNN：84.57%
- LSTM：91.00%

不同 eps 下的攻击成功率：



MIM 方法

初始化参数：

```
from momentum_iterative_method import momentum_iterative_method

mim_params = {"norm": np.inf, "eps": 20, "eps_iter": 1, "nb_iter": 1000, "clip_min": 0.0, "clip_max":
200.0, "sanity_checks": False}
```

生成对抗样本：

```
adv_x = momentum_iterative_method(logits_model, X_test, **mim_params)
adv_pred = model.predict(adv_x)
```

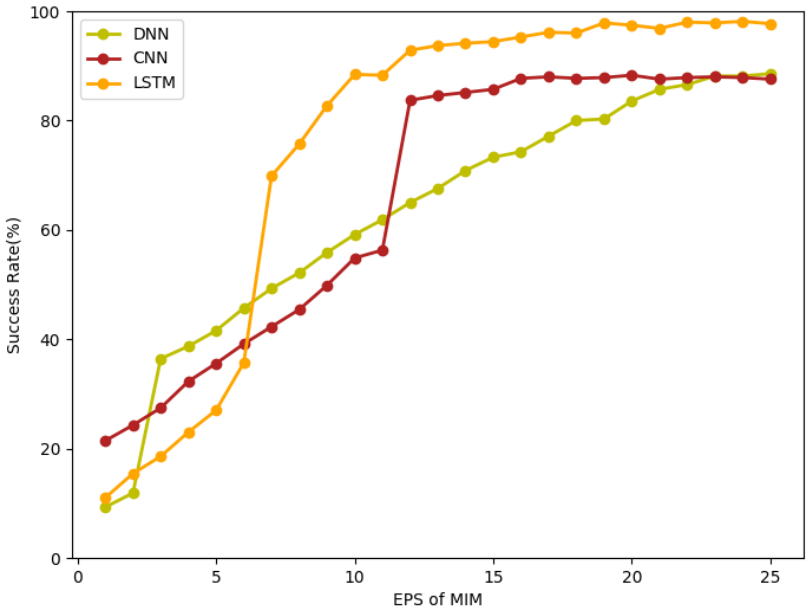
统计攻击成功率：

```
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

MIM 攻击成功率：

- DNN：83.57%
- CNN：88.29%
- LSTM：97.43%

不同 eps 下的攻击成功率：



CW 方法

初始化参数：

```
from carlini_wagner_l2 import carlini_wagner_l2

cw_params = {"clip_min": 0.0, "clip_max": 200.0, "initial_const": 1}
```

生成对抗样本：

```
adv_x = carlini_wagner_l2(logits_model, X_test, **cw_params)
adv_pred = model.predict(adv_x)
```

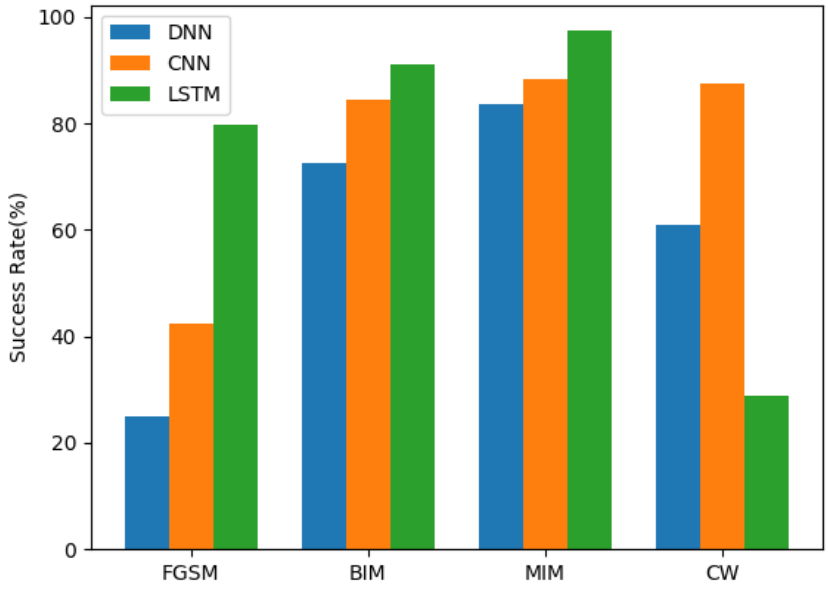
统计攻击成功率：

```
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

CW 攻击成功率：

- DNN：60.86%
- CNN：87.57%
- LSTM：28.86%

与前述三种方法的攻击成功率对比：



修正

上述实验成功率比实际值偏高，因为原始负样本中包含少量模型本身无法识别的样本。

```
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

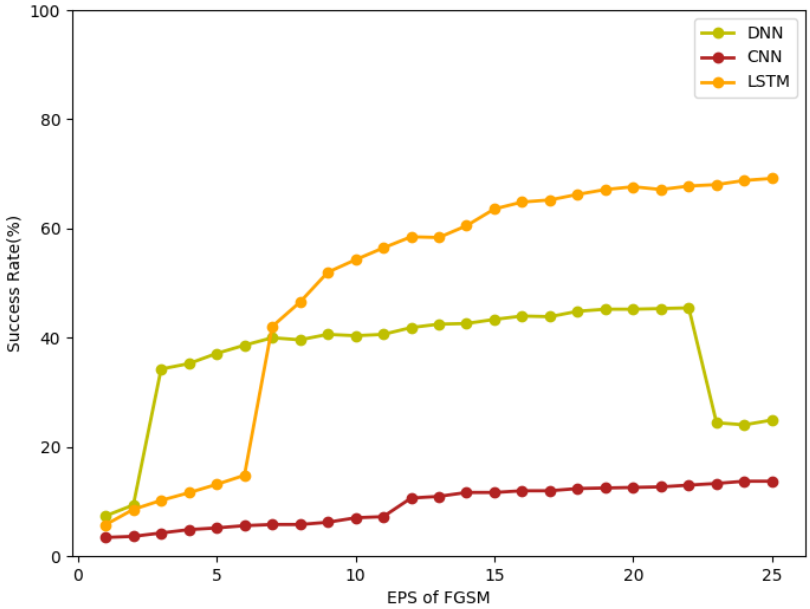
应修改为：

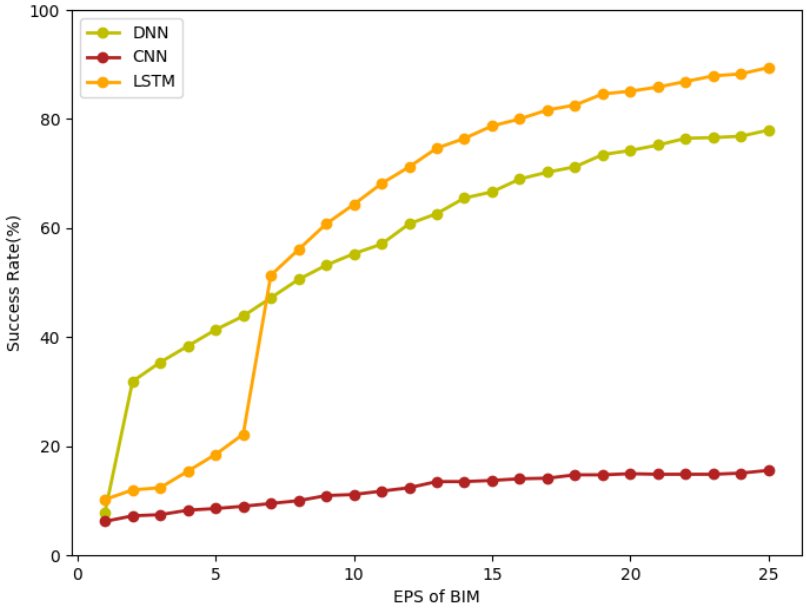
```
pred = model.predict(X_test)
count = 0
for i in range(700):
    if adv_pred[i][0] > 0.5 and pred[i][0] < 0.5:
        count += 1
print("Success rate: " + str(round(count/7.0, 2)) + "%")
```

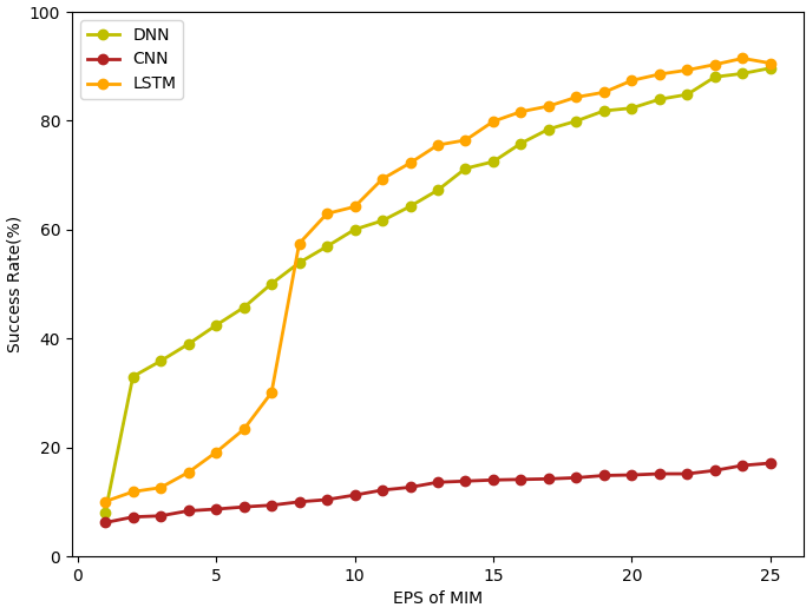
如果选择全部真实负样本进行实验，则攻击成功率会受到模型本身准确率的影响。

另外，验证攻击所使用的负样本集与训练集有重叠，应单独使用测试集中的负样本进行对抗攻击实验。

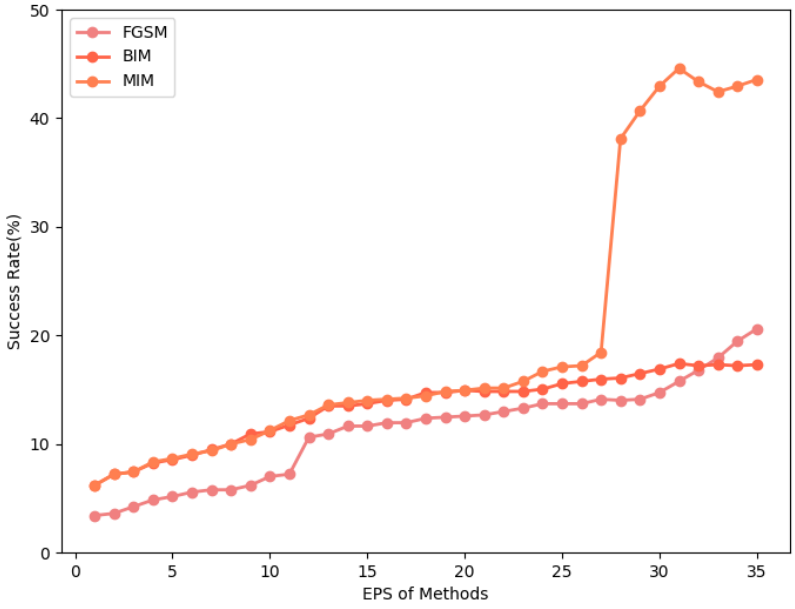
修正后的实验结果为：


DNN
CNN
LSTM
Success Rate(%)
EPS of FGSM

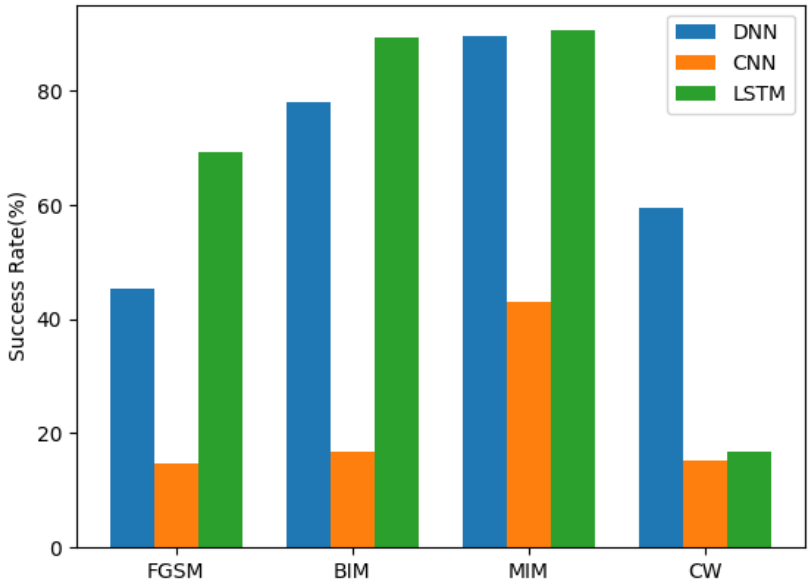

DNN
CNN
LSTM
Success Rate(%)
EPS of BIM



将 EPS 范围修改为 1-35，统计 CNN 的攻击成功率：



不同方法攻击成功率的对比：



改进的白盒方法

策略：更新对抗样本的过程中，不允许减小数据包的大小（相对于原始数据包）。

方法：

1. 经过固定迭代次数，修正对抗样本
2. 每次迭代中删除非法扰动，设置一个较大的 EPS 用于单次迭代，设置一个较小的 EPS 用于最终输出

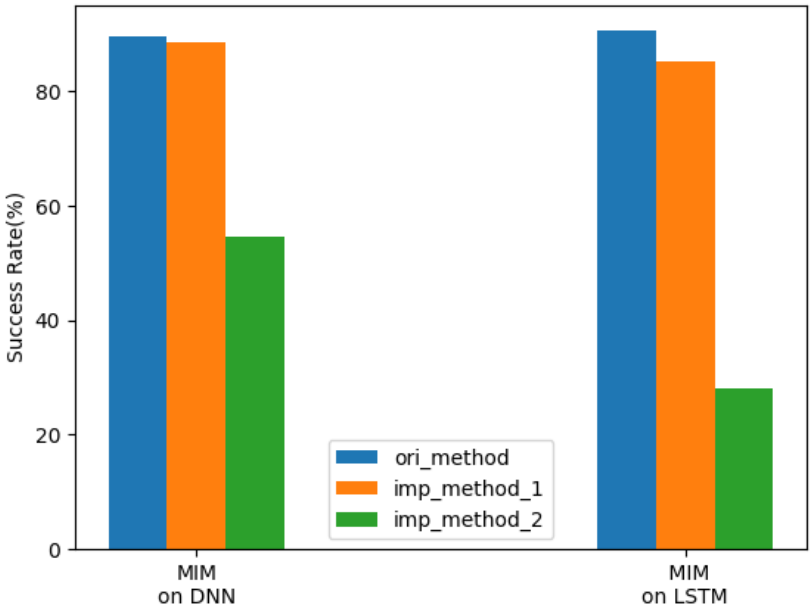
方法 1：（eps=25）

```
del_x = clip_eta(adv_x - x, norm, eps)
if i % 100 == 0:
    del_x = tf.clip_by_value(clip_eta(adv_x - x, norm, eps), 0, eps_iter)
adv_x = x + del_x
```

方法 2：（eps=25，eps_iter=21）

```
del_x = tf.clip_by_value(clip_eta(adv_x - x, norm, eps), 0, eps_iter)
adv_x = x + del_x
```

以 MIM 作为基础方法，实验结果如下：


Success Rate(%)
0
20
40
60
80
ori_method
imp_method_1
imp_method_2
MIM
on DNN
MIM
on LSTM