detector

December 16, 2021

1 Machine Learning in Cyber Security LAB3

zz3230 Zhe ZHANG

1.1 Prework

Import Libraries

[1]:
```
import h5py
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import os
import matplotlib.pyplot as plt
```

File routes

[2]:
```
### data_file
clean_test_file = './data/cl/test.h5'
clean_valid_file = './data/cl/valid.h5'
bad_test_file = './data/bd/bd_test.h5'
bad_valid_file = './data/bd/bd_valid.h5'

### model_file
bad_net_file = './models/bd_net.h5'
bad_net_weights = './models/bd_weights.h5'
```

[3]:
```
if os.path.isdir('./result')== False:
    os.mkdir('./result')
```

data loader

[4]:
```
### Load data
def data_loader(filepath):
    data = h5py.File(filepath, 'r')
    x_data = np.array(data['data'])
    y_data = np.array(data['label'])
    x_data = x_data.transpose((0,2,3,1))
```

```
    #x_data = data_preprocess(x_data)
    print(filepath + " loaded")
    return x_data, y_data
```

data processor

[5]:
```
def data_preprocess(x_data):
    return x_data/255
```

net loader

[6]:
```
### Load net model & weights
def net_loader(net_file):
    badNet = keras.models.load_model(net_file)
    print("model " + net_file + " loaded")

    return badNet
```

network weights loader

[7]:
```
def weights_loader(self, weights_file):
    self.load_weights(weights_file)
    print("weights " + weights_file + " for " + \
          "model " + net_file + " loaded")
    return self
```

evaluate backdoored network performance

[8]:
```
def eval_origin(bd_model, cl_x_test, cl_y_test, bd_x_test, bd_y_test):

    cl_label_p = np.argmax(bd_model.predict(cl_x_test), axis=1)
    clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test))*100
    print('Clean Classification accuracy:', clean_accuracy)

    bd_label_p = np.argmax(bd_model.predict(bd_x_test), axis=1)
    attack_rate = np.mean(np.equal(bd_label_p, bd_y_test))*100
    print('Attack Success Rate:', attack_rate)
    return clean_accuracy
```

[9]:
```
X_clean_valid, y_clean_valid = data_loader(clean_valid_file)
X_bad_valid, y_bad_valid = data_loader(bad_valid_file)
```

```
./data/cl/valid.h5 loaded
./data/bd/bd_valid.h5 loaded
```

[10]:
```
bd_model = net_loader(bad_net_file)
origin_acc = eval_origin(bd_model, X_clean_valid, y_clean_valid, X_bad_valid,␣
 ↪y_bad_valid)
```

```
model ./models/bd_net.h5 loaded
```

```
Clean Classification accuracy: 98.64899974019225
Attack Success Rate: 100.0
```

1.2 Prune Channels

You will design G using the pruning defense that we discussed in class. That is, you will prune the last pooling layer of BadNet B (the layer just before the FC layers) by removing one channel at a time from that layer. Channels should be removed in decreasing order of average activation values over the entire validation set. Every time you prune a channel, you will measure the new validation accuracy of the new pruned badnet. You will stop pruning once the validation accuracy drops atleast X% below the original accuracy. This will be your new network B'

show the structure of bd_net

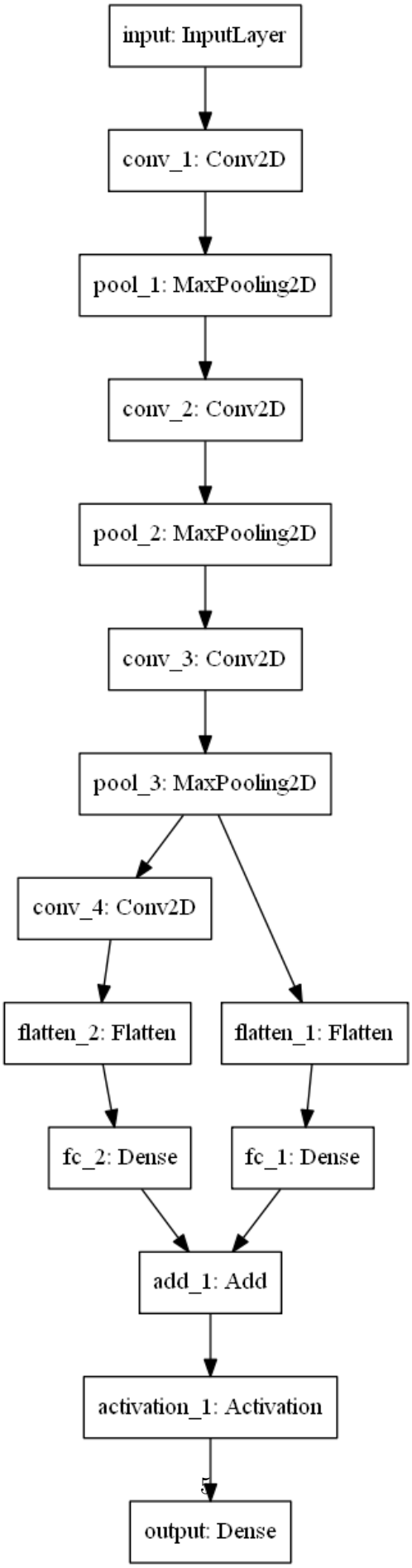
[11]:
```
bd_model.summary()
```

```
Model: "model_1"
_________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
=================
input (InputLayer)           [(None, 55, 47, 3)]       0
_________________________________________________________________________________
_________________
conv_1 (Conv2D)              (None, 52, 44, 20)        980         input[0][0]
_________________________________________________________________________________
_________________
pool_1 (MaxPooling2D)        (None, 26, 22, 20)        0           conv_1[0][0]
_________________________________________________________________________________
_________________
conv_2 (Conv2D)              (None, 24, 20, 40)        7240        pool_1[0][0]
_________________________________________________________________________________
_________________
pool_2 (MaxPooling2D)        (None, 12, 10, 40)        0           conv_2[0][0]
_________________________________________________________________________________
_________________
conv_3 (Conv2D)              (None, 10, 8, 60)         21660       pool_2[0][0]
_________________________________________________________________________________
_________________
pool_3 (MaxPooling2D)        (None, 5, 4, 60)          0           conv_3[0][0]
_________________________________________________________________________________
_________________
conv_4 (Conv2D)              (None, 4, 3, 80)          19280       pool_3[0][0]
_________________________________________________________________________________
_________________
flatten_1 (Flatten)          (None, 1200)              0           pool_3[0][0]
_________________________________________________________________________________
_________________
```

```
flatten_2 (Flatten)             (None, 960)          0           conv_4[0][0]
__________________________________________________________________________________
__________________
fc_1 (Dense)                    (None, 160)          192160      flatten_1[0][0]
__________________________________________________________________________________
__________________
fc_2 (Dense)                    (None, 160)          153760      flatten_2[0][0]
__________________________________________________________________________________
__________________
add_1 (Add)                     (None, 160)          0           fc_1[0][0]
                                                                 fc_2[0][0]
__________________________________________________________________________________
__________________
activation_1 (Activation)       (None, 160)          0           add_1[0][0]
__________________________________________________________________________________
__________________
output (Dense)                  (None, 1283)         206563
activation_1[0][0]
==================================================================================
==================
Total params: 601,643
Trainable params: 601,643
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

[12]:
```
keras.utils.plot_model(bd_model)
```

[12]:


input: InputLayer
conv_1: Conv2D
pool_1: MaxPooling2D
conv_2: Conv2D
pool_2: MaxPooling2D
conv_3: Conv2D
pool_3: MaxPooling2D
conv_4: Conv2D
flatten_2: Flatten
flatten_1: Flatten
fc_2: Dense
fc_1: Dense
add_1: Add
activation_1: Activation
output: Dense

The specific layer we would prune is the Conv2D layer named `conv_3'

Copy the backdoored model and find the specific layer to prune

load network and weights

[13]:
```
prune_model = keras.models.clone_model(bd_model)
prune_model.load_weights(bad_net_weights)
```

prune the last pooling layer of BadNet B (the layer just before the FC layers) by removing one channel at a time from that layer. Channels should be removed in **increasing order** of average activation values over the entire validation set.

[14]:
```
activation_output = prune_model.layers[6].output
avg_activation = np.mean(keras.models.Model(inputs=prune_model.input,␣
 ↪outputs=activation_output).predict(X_clean_valid),axis=(0,1,2))
sorted_avg_act_channel = np.argsort(avg_activation)
```

show the increasing order of activation values

[15]:
```
print("Increasing order of average activation values")
print(avg_activation[sorted_avg_act_channel])
```

```
Increasing order of average activation values
[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 3.0290761e-03 6.2408661e-03 1.3321567e-02 1.5006668e-02
 4.3979675e-02 8.3539173e-02 1.8378231e-01 2.4381575e-01 4.2763174e-01
 5.0732863e-01 5.3079778e-01 5.7658589e-01 8.5787797e-01 1.0589781e+00
 1.5654888e+00 1.6352932e+00 1.8540378e+00 2.0289588e+00 2.1106052e+00
 2.1980376e+00 3.6191154e+00 4.1488924e+00 4.8440871e+00 4.8648086e+00
 5.0868411e+00 5.1451392e+00 5.3690357e+00 6.2038713e+00 8.2229824e+00]
```

Every time you prune a channel, you will measure the new validation accuracy of the new pruned badnet. You will stop pruning once the validation accuracy drops **atleast X% below the original accuracy**.

[16]:
```
print("The original accuracy is  " + str(origin_acc))
```

```
The original accuracy is  98.64899974019225
```

[17]:
```
X = np.array([2,4,10,30])
```

We are going to prune the channels, which means set the W_h and b_h to be 0 in each single channel of the ``**conv_3**'' layer

Prune Channel in increasing order

[18]:
```
def␣
↪prune_channel(prune_model,X,X_clean_valid,y_clean_valid,X_bad_valid,y_bad_valid,save=False)
↪

    W_h = prune_model.layers[5].get_weights()[0]
    b_h = prune_model.layers[5].get_weights()[1]

    acc_ori = np.mean(np.equal(np.argmax(prune_model.predict(X_clean_valid),␣
↪axis=1), y_clean_valid))*100
    atk_ori = np.mean(np.equal(np.argmax(prune_model.predict(X_bad_valid),␣
↪axis=1), y_bad_valid))*100

    acc = []
    atk = []
    acc_tmp = 0
    atk_tmp = 0

    status = np.zeros(len(X))
    prune_num = np.zeros(len(X))

    for i in range(len(sorted_avg_act_channel)):

        ch = sorted_avg_act_channel[i]

        ### set and update w_h, b_h
        W_h[:,:,:,ch] = 0
        b_h[ch] = 0
        prune_model.layers[5].set_weights([W_h,b_h])

        ### predict and get the acc of the new network
        if abs(avg_activation[ch] - 0) < 0.0000001:
            acc_tmp = acc_ori
            atk_tmp = atk_ori
        else:
            acc_tmp = np.mean(np.equal(np.argmax(prune_model.
↪predict(X_clean_valid), axis=1), y_clean_valid))*100
            atk_tmp = np.mean(np.equal(np.argmax(prune_model.
↪predict(X_bad_valid), axis=1), y_bad_valid))*100

        acc.append(acc_tmp)
        atk.append(atk_tmp)
        print(str(i+1)+' channels pruned, acc = '+str(acc_tmp)+', atk =␣
↪'+str(atk_tmp)+'  X = '+ str(acc_ori - acc_tmp))

        ### test and save the model if acc is X below the orignal acc
```

```
        if save == True:
            for j in range(len(X)):
                if acc_ori - acc_tmp > X[j] and status[j] == 0:
                    status[j] = 1
                    prune_num[j] = i+1
                    prune_model.save('./result/
↪'+'pruning_channel_model_acc_decrease_by_'+str(X[j])+'%.h5')
                    print('pruning channel model accuracy decrease by␣
↪'+str(X[j])+' saved at prune channels of '+str( prune_num[j]))

        keras.backend.clear_session()

    return acc, atk, prune_num
```

[19]:
```
acc, atk, prune_num =␣
↪prune_channel(prune_model,X,X_clean_valid,y_clean_valid,X_bad_valid,y_bad_valid,save=True)
```

```
1 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
2 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
3 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
4 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
5 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
6 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
7 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
8 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
9 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
10 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
11 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
12 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
13 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
14 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
15 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
16 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
17 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
18 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
19 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
20 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
21 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
22 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
23 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
24 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
25 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
26 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
27 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
28 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
29 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
30 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
```

```
31 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
32 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
33 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
34 channels pruned, acc = 98.64033948211657, atk = 100.0  X =
0.00866025807567894
35 channels pruned, acc = 98.64033948211657, atk = 100.0  X =
0.00866025807567894
36 channels pruned, acc = 98.63167922404088, atk = 100.0  X =
0.01732051615137209
37 channels pruned, acc = 98.65765999826795, atk = 100.0  X =
-0.00866025807569315
38 channels pruned, acc = 98.64899974019225, atk = 100.0  X = 0.0
39 channels pruned, acc = 98.6056984498138, atk = 100.0  X = 0.04330129037845154
40 channels pruned, acc = 98.57105741751104, atk = 100.0  X =
0.07794232268120993
41 channels pruned, acc = 98.53641638520828, atk = 100.0  X =
0.11258335498396832
42 channels pruned, acc = 98.19000606218066, atk = 100.0  X =
0.45899367801159485
43 channels pruned, acc = 97.65307006148784, atk = 100.0  X = 0.9959296787044138
44 channels pruned, acc = 97.50584567420108, atk = 100.0  X = 1.143154065991169
45 channels pruned, acc = 95.75647354291158, atk = 100.0  X = 2.8925261972806737
pruning channel model accuracy decrease by 2 saved at prune channels of 45.0
46 channels pruned, acc = 95.20221702606739, atk = 99.9913397419243  X =
3.446782714124865
47 channels pruned, acc = 94.7172425738287, atk = 99.9913397419243  X =
3.9317571663635533
48 channels pruned, acc = 92.09318437689443, atk = 99.9913397419243  X =
6.555815363297825
pruning channel model accuracy decrease by 4 saved at prune channels of 48.0
49 channels pruned, acc = 91.49562656967177, atk = 99.9913397419243  X =
7.1533731705204815
50 channels pruned, acc = 91.01931237550879, atk = 99.98267948384861  X =
7.629687364683463
51 channels pruned, acc = 89.17467740538669, atk = 80.73958603966398  X =
9.474322334805564
52 channels pruned, acc = 84.43751623798389, atk = 77.015675067117  X =
14.211483502208367
pruning channel model accuracy decrease by 10 saved at prune channels of 52.0
53 channels pruned, acc = 76.48739932449988, atk = 35.71490430414826  X =
22.161600415692376
54 channels pruned, acc = 54.8627349095003, atk = 6.954187234779596  X =
43.786264830691955
pruning channel model accuracy decrease by 30 saved at prune channels of 54.0
55 channels pruned, acc = 27.08928726076037, atk = 0.4243526457088421  X =
71.55971247943188
56 channels pruned, acc = 13.87373343725643, atk = 0.0  X = 84.77526630293582
57 channels pruned, acc = 7.101411622066338, atk = 0.0  X = 91.54758811812592
```

```
58 channels pruned, acc = 1.5501861955486274, atk = 0.0  X = 97.09881354464362
59 channels pruned, acc = 0.7188014202823244, atk = 0.0  X = 97.93019831990993
60 channels pruned, acc = 0.0779423226812159, atk = 0.0  X = 98.57105741751104
```

1.3 Good Network

What you must output is G a ``repaired'' BadNet. G has N+1 classes, and given unseen test input, it must:

1. Output the correct class if the test input is clean. The correct class will be in [1,N].
2. Output class N+1 if the input is backdoored

For each test input, you will run it through both B and B'. If the classification outputs are the same, i.e., class i, you will output class i. If they differ you will output N+1

Generate good network

[20]:
```
class good_network(keras.Model):

    def __init__(self, B, B_pruned):
        super(good_network,self).__init__()
        self.B = B
        self.B_pruned = B_pruned

    def predict(self, X):
        yhat = np.argmax(self.B.predict(X),axis=1)
        yhat_pruned = np.argmax(self.B_pruned.predict(X),axis=1)
        yhat[np.where(yhat!=yhat_pruned)] = 1283

        return yhat
```

Combine network B and network $B^{'}$ to be the repaired network G

[21]:
```
repaired_model_2p = good_network(bd_model, net_loader('./result/
 ↪pruning_channel_model_acc_decrease_by_2%.h5'))
repaired_model_4p = good_network(bd_model, net_loader('./result/
 ↪pruning_channel_model_acc_decrease_by_4%.h5'))
repaired_model_10p = good_network(bd_model, net_loader('./result/
 ↪pruning_channel_model_acc_decrease_by_10%.h5'))
repaired_model_30p = good_network(bd_model, net_loader('./result/
 ↪pruning_channel_model_acc_decrease_by_30%.h5'))
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
model ./result/pruning_channel_model_acc_decrease_by_2%.h5 loaded
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
model ./result/pruning_channel_model_acc_decrease_by_4%.h5 loaded
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
model ./result/pruning_channel_model_acc_decrease_by_10%.h5 loaded
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
model ./result/pruning_channel_model_acc_decrease_by_30%.h5 loaded
```

1.4 Evaluation

However, the repaired network G is not a complete model but does have ability to classify, SO, we rewrite the eval.py function here

Evaluation function for good network

[22]:
```
def eval_repaired(model, X_clean_test,y_clean_test,X_bad_test,y_bad_test):
    cl_label_p = model.predict(X_clean_test)
    clean_accuracy = np.mean(np.equal(cl_label_p, y_clean_test))*100
    print('Clean Classification accuracy:', clean_accuracy)

    bd_label_p = model.predict(X_bad_test)
    attack_rate = np.mean(np.equal(bd_label_p, y_bad_test))*100
    #print(bd_label_p)
    #print(y_bad_test)
    print('Attack Success Rate:', attack_rate)
    return clean_accuracy
```

load test data

[23]:
```
X_clean_test, y_clean_test = data_loader(clean_test_file)
X_bad_test, y_bad_test = data_loader(bad_test_file)
```

```
./data/cl/test.h5 loaded
./data/bd/bd_test.h5 loaded
```

We will compare the performance among **original backdoored network** and **repaired networks**

[24]:
```
print("backdoored network on test dataset")
eval_origin(bd_model, X_clean_test,y_clean_test,X_bad_test,y_bad_test)
print("repaired network, channels pruned with a 2% drop of accuracy, on test␣
 ↪dataset")
eval_repaired(repaired_model_2p,␣
 ↪X_clean_test,y_clean_test,X_bad_test,y_bad_test)
print("repaired network, channels pruned with a 4% drop of accuracy, on test␣
 ↪dataset")
eval_repaired(repaired_model_4p,␣
 ↪X_clean_test,y_clean_test,X_bad_test,y_bad_test)
print("repaired network, channels pruned with a 10% drop of accuracy, on test␣
 ↪dataset")
```

```
eval_repaired(repaired_model_10p,␣
 ↪X_clean_test,y_clean_test,X_bad_test,y_bad_test)
print("repaired network, channels pruned with a 30% drop of accuracy, on test␣
 ↪dataset")
eval_repaired(repaired_model_30p,␣
 ↪X_clean_test,y_clean_test,X_bad_test,y_bad_test)
```

```
backdoored network on test dataset
Clean Classification accuracy: 98.62042088854248
Attack Success Rate: 100.0
repaired network, channels pruned with a 2% drop of accuracy, on test dataset
Clean Classification accuracy: 95.74434918160561
Attack Success Rate: 100.0
repaired network, channels pruned with a 4% drop of accuracy, on test dataset
Clean Classification accuracy: 92.1278254091972
Attack Success Rate: 99.98441153546376
repaired network, channels pruned with a 10% drop of accuracy, on test dataset
Clean Classification accuracy: 84.3335931410756
Attack Success Rate: 77.20966484801247
repaired network, channels pruned with a 30% drop of accuracy, on test dataset
Clean Classification accuracy: 54.67653936087296
Attack Success Rate: 6.96024941543258
```

[24]:
```
54.67653936087296
```

Plot the accuracy on **clean test data** and the attack success rate (on **backdoored test data**) as a function of the fraction of channels pruned

[25]:
```
prune_model_test = keras.models.clone_model(bd_model)
prune_model_test.load_weights(bad_net_weights)
acc_test, atk_test, prune_num =␣
 ↪prune_channel(prune_model_test,X,X_clean_test,y_clean_test,X_bad_test,y_bad_test,save=False
```

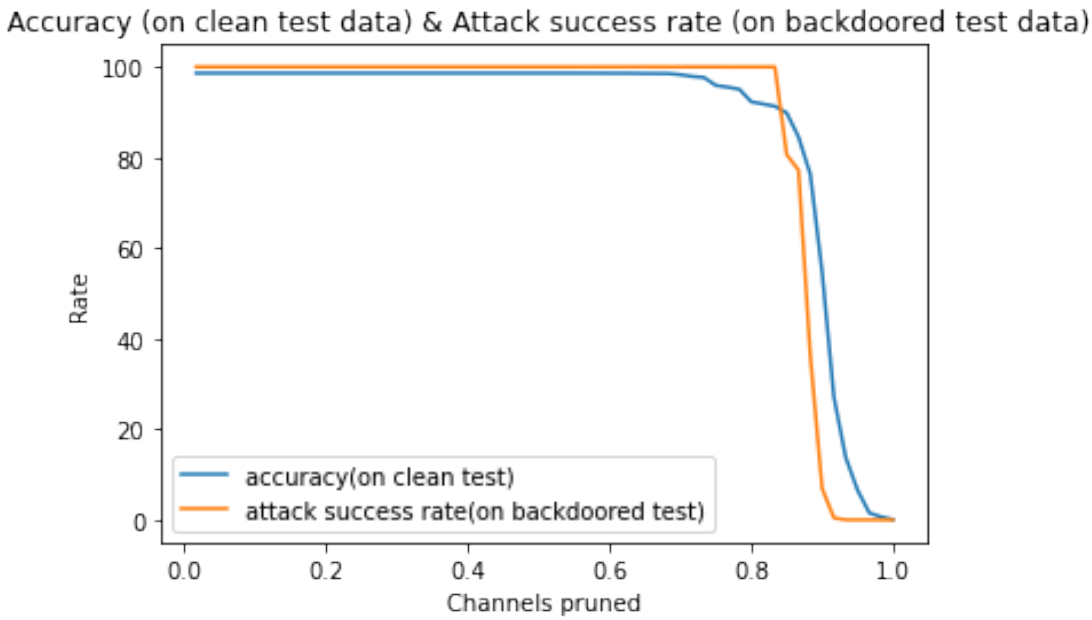
```
1 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
2 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
3 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
4 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
5 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
6 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
7 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
8 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
9 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
10 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
11 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
12 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
13 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
14 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
15 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
16 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
```

```
17 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
18 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
19 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
20 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
21 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
22 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
23 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
24 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
25 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
26 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
27 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
28 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
29 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
30 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
31 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
32 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
33 channels pruned, acc = 98.62042088854248, atk = 100.0  X = 0.0
34 channels pruned, acc = 98.61262665627436, atk = 100.0  X =
0.007794232268125256
35 channels pruned, acc = 98.61262665627436, atk = 100.0  X =
0.007794232268125256
36 channels pruned, acc = 98.60483242400623, atk = 100.0  X =
0.015588464536250513
37 channels pruned, acc = 98.60483242400623, atk = 100.0  X =
0.015588464536250513
38 channels pruned, acc = 98.58924395947, atk = 100.0  X = 0.031176929072486814
39 channels pruned, acc = 98.55027279812938, atk = 100.0  X =
0.07014809041309888
40 channels pruned, acc = 98.53468433359313, atk = 100.0  X = 0.0857365549493494
41 channels pruned, acc = 98.52689010132501, atk = 100.0  X =
0.09353078721747465
42 channels pruned, acc = 98.269680436477, atk = 100.0  X = 0.3507404520654802
43 channels pruned, acc = 97.88776305533905, atk = 100.0  X = 0.732657833203433
44 channels pruned, acc = 97.66173031956352, atk = 100.0  X = 0.958690568978966
45 channels pruned, acc = 95.90023382696803, atk = 100.0  X = 2.7201870615744497
46 channels pruned, acc = 95.5261106780982, atk = 99.97661730319564  X =
3.0943102104442772
47 channels pruned, acc = 95.0584567420109, atk = 99.98441153546376  X =
3.5619641465315794
48 channels pruned, acc = 92.29150428682775, atk = 99.98441153546376  X =
6.328916601714738
49 channels pruned, acc = 91.8082618862042, atk = 99.98441153546376  X =
6.8121590023382765
50 channels pruned, acc = 91.30943102104443, atk = 99.97661730319564  X =
7.310989867498051
51 channels pruned, acc = 89.84411535463757, atk = 80.6469212782541  X =
8.776305533904917
52 channels pruned, acc = 84.54403741231489, atk = 77.20966484801247  X =
```

```
14.07638347622759
53 channels pruned, acc = 76.30553390491036, atk = 36.26656274356976  X =
22.31488698363212
54 channels pruned, acc = 54.762275915822286, atk = 6.96024941543258  X =
43.8581449727202
55 channels pruned, acc = 27.10054559625877, atk = 0.4208885424785659  X =
71.51987529228371
56 channels pruned, acc = 13.756819953234606, atk = 0.0  X = 84.86360093530787
57 channels pruned, acc = 6.570537802026501, atk = 0.0  X = 92.04988308651598
58 channels pruned, acc = 1.5198752922837102, atk = 0.0  X = 97.10054559625877
59 channels pruned, acc = 0.646921278254092, atk = 0.0  X = 97.97349961028839
60 channels pruned, acc = 0.0779423226812159, atk = 0.0  X = 98.54247856586127
```

[31]:
```
# the x axiel is actually from 1 to 60 and divide 60
x_axis = np.arange(1,61)/60
plt.title(" Accuracy (on clean test data) & Attack success rate (on backdoored␣
 ↪test data)")
plt.xlabel("Channels pruned")
plt.ylabel("Rate")
plt.plot(x_axis,acc_test)
plt.plot(x_axis,atk_test)
plt.legend(['accuracy(on clean test)','attack success rate(on backdoored␣
 ↪test)'])
```

[31]: <matplotlib.legend.Legend at 0x2194c73e3c8>



As **Lab3 report** and **readme file**, please read

```
'./README.md'
```