ECE-GY 9163
Machine Learning For Cyber Security

By
Prof. Siddharth Garg

Project Report

Rahul Chinthala
rc4080

Anand Inguva
api226

Rajvi Gandhi
rbg336

Urvi Rathod
ubr203

Machine Learning for Cyber Security

Approach:

Having been tasked with detecting adversarial inputs and repairing the bad nets, we tried out a couple of methods for the same, starting out with the easy to implement STRIP research paper. In essence, it states that when an image is perturbed by linear blending with an image, and run through a model. The entropy of the modified image is lower in the case where the original image is perturbed. However , we weren't precisely able to set the boundary value based on the mean and standard deviation of entropy values of clean inputs, thus we decided to search for other decisive methods. We then tried the Spectral signature method, however, we realized that the spectral signature method is for the case when trained data is just poisoned but not trained by the adversary. Also, the method requires us to have knowledge of epsilon, the percentage of label that’s poisoned. Since, we don’t have access to epsilon so we moved onto the next method. We finally decided to implement Fine Pruning, where we figured out the neurons in the badnet that were getting activated by clean data, purged the rest of the neurons by setting the weights to zero, retrained the model using using clean data, added additional neuron to the last layer of the model, and trained it to recognize poisoned data. We then ran inputs through the original badnet and the produced goodnet. The inputs whose predictions changed have been classified as adversarial inputs.

Explanation of Code

Libraries used : We used tensorflow keras as the primary tool for modelling the neural network.

Functions:

- **dataloader :** Loads data from the given file path , and returns feature and target columns.
- **data_preprocess :** Normalize the images , by dividing by 255.
- **get_model :** loads the model from the file directory and returns it.
- **get_accuracy :** Takes model , features , and corresponding target values as inputs , and returns model accuracy and predicted labels.
- **get_con_layer_indices :** Takes model as the input and returns the indices of the conv layers.
- **train_model** : Takes model, features and corresponding labels as inputs , and trains the model. We use sparse categorical cross entropy as the loss function, Adam optimizer , batch size of 64 , validation split as 0.2 , and 10 epochs.

- **pruning** : Takes features , layer number to be pruned , model to be pruned, and pruning percentage as inputs. The function calculates the activation values for each channel of the given layer of the model corresponding to the given input, and calculates the average activation value across these channels. We consider 100 inputs to calculate average activations here.It then sorts the channels by the mean activation value, and stores the channel numbers in sorted_indices variable. It then selects the top channels based on the pruning percentage , and sets the weights of the rest of the channels to zero. It then calculates the accuracy of the pruned model on poisoned and clean data , returning the pruned model , clean data accuracy, attack accuracy.We ran through all combinations of pruning percentages(10% to 90%), and layer indices to figure out the layer number and pruning percentage that resulted in lowest attack accuracy , and clean accuracy greater 55%. This is shown in the ipynb file as " Model after pruning , before tuning". The layer and pruning percent wise clean and attack accuracy are stored in the "percent_acc" dictionary.
- Having selected the pruned the models , we fine tuned them by training the model using clean validation data, which resulted in clean accuracy greater than about 93-95% and attack accuracy below 5% for all the four models.
- However, adding an additional neuron to the last year, in order for the goodnet to recognize and label the poisoned data resulted in reduction in clean data accuracy to about 87-90%.
- We could , however, do away with adding an additional neuron to recognize the poisoned output as we could simply run the input on the original badnet and fine pruned net , then compare both the predictions and declare the input as poisoned if the predictions don't match. Doing this would have aided us in detecting poisoned inputs and while still having about 93% accuracy on clean data. Consider , the multi trigger, Multi target badnet for example:
- **Original Multi trigger, Multi target Badnet**
  Clean accuracy is 96.26742876937733
  Attack accuracy is 94.29072486360094
- **After Fine Pruning**
  Clean accuracy is 93.981120637395
  Attack accuracy is 5.8554169914263445
- **After adding extra neuron in last layer, and retraining to detect poisoned data**
  Clean accuracy is 87.25210011258335
  Poisoned data detection accuracy is 99.25630033775008
- In order to obtain the final good net for a given badnet, go to cell 3 in the .ipynb file and change the Poison_data_filename, model_filename, and save_path_name. Then run all cells to have the good_nets weights in the corresponding save path.