CIS590/890 DL - Assignment 2

1 Assignment Goals

In this assignment, you will train and evaluate neural network models for identifying Android malware. In the process, you will gain experience with a deep learning framework, PyTorch or TensorFlow 2.0, and also with TensorBoard. You will also experiment with training the model on CPU and GPU (potentially, in the cloud), and compare the running times.

2 Note

Useful links and PyTorch/TensorFlow tutorials for this assignment are provided on Canvas. Questions and discussions on Canvas are also encouraged.

3 Data Used

The data you will use in this assignment represents Android apps that are labeled as benign or malicious. The goal is to build models that can identify malicious apps (or malware), so that they can be prevented from entering the market. It is estimated that the ratio of malware to benign apps is close to 1:100. Two sets of apps are provided.

First, we provide small, balanced train and test datasets (i.e, ratio is 1:1) that you can use to develop your code. These datasets are provided as csv files on Canvas (`AndroidAppsTrainSmall.csv` and `AndroidAppsTestSmall.csv`). Each app in the train and test files is represented using 471 binary (0/1) features, which denote the presence/absence of various permissions, intent actions, discriminative APIs, obfuscation signatures, and native code signatures. For simplicity sake, you can assume that the names of the features are $f0, f2, \cdots, f470$. The last column of an instance/app ($f471$) corresponds to the class label y, which takes values 0 (benign) or 1 (malware).

In addition to the small train/test datasets described above, we also provide larger datasets (with imbalance ratio 1:10) to make the learning problem more challenging. The larger train and test datasets are also provided as csv files on Canvas (`AndroidAppsTrainLarge.csv` and `AndroidAppsTestLarge.csv`, respectively). The app representation is the same as above.

4 Tasks

Perform the following tasks:

1. Create a custom TensorFlow/PyTorch Dataset.
2. Use TensorFlow keras or PyTorch nn to create two two-layer fully-connected neural networks. The networks have an input dimension of N (given by the number of features in an app), a hidden layer dimension of H (your choice), and perform classification over 2 classes.

   The first network uses a linear activation after the first fully connected layer, and the sigmoid function to compute the output. In other words, the first network has the following architecture (where FC means "fully connected layer"): `input - FC - linear - FC - sigmoid`.

   The second network uses the ReLU activation function after the first layer, and the sigmoid to compute the output. Thus, the architecture of the second network is: `input - FC - ReLU - FC - sigmoid`. The `binary_crossentropy` is used to compute the loss in both cases. Use a subset of the training data as validation to perform hyperparamter tuning.
3. Log your models' training and use TensorBoard to plot learning curves that show the variation of the loss/accuracy with the number of epochs (fixed) for both training and validations sets, for three different values of H, for both networks. The learning rate can also be fixed, unless you'd like to experiment with two parameters together.
4. Identify your best model according to the validation data, and use it to evaluate the performance on the test dataset in terms of accuracy. In addition, report the precision, recall and F-measure on the malware class.
5. Train the non-linear model on the larger dataset, and evaluate it on the corresponding test dataset. Report the precision, recall and F-measure on the malware class.
6. Use both the large and the small datasets to compare the model training times on CPU versus GPU.

5 What to submit

Submit a Jupiter Notebook containing your code and results/plots, or python code together with a report file showing the results obtained for different task.