Trojan-Sleuth (Version 2.0)

1. Background

The Trojan-Sleuth project is a competitor in IARPA's TrojAI competition. The goal of this competition is to accurately assess whether a pre-trained neural network model has been infected with a Trojan (i.e., backdoor) during its training process. The current version deals with text data, while Trojan Sleuth 1.0 dealt with image data[1]. The competition site can be found here: https://www.nist.gov/itl/ssd/trojai and the trained models can be found here: https://pages.nist.gov/trojai/docs/data.html

2. Intuition for Trojan-Sleuth

A Trojan-infected neural network model is one that has been trained such that a small modification to a sample will cause a misclassification to a specified class. This specified class is known as the target class. The modification to the sample is known as the trigger. In Rounds 5 and 6, we are dealing with binary sentiment classification models, where a trigger would incorrectly flip the predicted sentiment. The sentiment classifiers are trained on sequence-level word embeddings which encode the sentence's semantic meaning. Our original proposal was centered around the concept that the geometrical properties of gradients can be an indicator of Trojan behavior. Continuing along this line of thought, our intuition suggests that if we perturb the embeddings, we can cause geometric anomalies in the gradients with respect to those embeddings, which can then be used for Trojan identification.

3. Methodology

NIST provides labeled data in the form of IMDB and Amazon reviews.[2] The sentiment classifiers are LSTM and GRU models, of which half are clean and the other half have been infected with a Trojan. Trojan triggers can either be a character, word, or phrase inserted into the review. The overall strategy of Trojan-Sleuth is to use geometric properties of gradients to distinguish clean from Trojan-infected models. Based on preliminary research, when the trigger is added to samples belonging to a Trojan-infected model, their gradients experience geometric anomalies. Because the trigger is unknown at runtime, the samples' embeddings are perturbed to roughly emulate the effect of a trigger. The gradients of the model with respect to the embeddings are then calculated, and geometric properties are measured by taking the mean and standard deviation. Training a classifier on these statistics yields a reliable way to detect Trojan-infected models.

[1] https://github.com/perspectalabs/Iliad-TrojanSleuth/blob/main/README.pdf
[2] https://pages.nist.gov/trojai/docs/data.html#round-5

3.1 Feature generation (feature_generation.py)

3.1.1 Pre-Processing

Each text sample must be tokenized and converted into a numerical embedding before being fed into the sentiment classification model. The type of tokenizer and embedding is known at runtime to ensure it appropriately corresponds with the model. The three types of embeddings used in Round 5 are BERT, DistilBERT, and GPT-2. The sentiment classification models are only trained on the sequence-level embeddings, rather than the individual words' embeddings. The sequence embedding should hold semantic information of the whole text, in a numeric 768-dimensional vector.

3.1.2 Computing gradients and their statistics

For each class, 10 example reviews are taken, and for each one the embedding is computed. For each of these embeddings, 5 perturbations are created. The perturbations are created by adding random noise normally distributed around 0 with a standard deviation equal to the embedding's standard deviation. Because there are 2 classes, this amounts to a set of 100 perturbed embedding vectors. Each is scaled to mean 0 and unit variance, and fed through the sentiment classifier. The Jacobians of the 1st and 2nd logits, and gradient of the loss with respect to the input is taken. This produces 3 sets of features, each one with 768 elements since that is the size of the input vector. After concatenating these features, there is a 100 x 2304 array of Jacobians and gradients. Finally, the means and standard deviations are taken row-wise, to produce a 1 x 4608 -dimension feature vector.

3.1.3 Code

To generate the feature vectors for each of the 1,656 models supplied by NIST, run the following code in an environment that meets the dependency requirements

```
python3 feature_generation.py
```

This code will produce the features described in 3.1.2 and save them locally to round5.csv

The majority of the code is held within the function get_model(). This function takes two parameters:

- *num_examples:* the number of training examples to use per class. By default, this is set to 10.
- *num_perturb:* the number of perturbations to produce per example. By default, this is set to 5.

3.2 Training the classifier (svm_training.py)

Once features have been generated from the training data and have been saved as round5.csv, SVMs are used to learn a Trojan-detection classifier. Utilizing sample-wise bagging, six SVMs with RBF kernels are each trained on 5/6 of the training data and ensembled together. Their hyper-parameters, C and gamma, are learned through cross-validation. The data is split into a training and test set to get a sense of its generalizability, and some metrics are printed. The final classifier is saved as round5.joblib.

3.2.1 Code

The Trojan-detection classifier is trained by running the following code

```
python3 svm_training.py
```

3.3 Evaluation (trojan_detector_round5.py)

trojan_detector_round5.py utilizes the entire method described in this document to evaluate a single model. It pre-processes given text data, obtains features, and runs those features through a pre-trained SVM model to calculate the probability the model is infected with a Trojan.

3.3.1 Code

To run using the default arguments

```
python3 trojan_detector_round5.py
```

trojan_detector_round5.py takes many optional parameters to specify which model you are evaluating

- model_filepath: the file containing the model you are assessing.
- cls_token_is_first: whether the CLS token is the first token in the input sequence (True for BERT models and false for GPT models)
- tokenizer_filepath: the file containing the tokenizer for converting input text into tokens
- embedding_filepath: the file containing the embedding model for converting tokens into a numerical embedding vector
- results_filepath: the file you will print your final probability to.
- examples_dirpath: the directory containing example text in the form of reviews
- scratch: a directory to perform scratch work in (it is not used in this version of Trojan-Sleuth).

Example output should look like this:

```
Trojan Probability: 0.9605976828221922
```

4.0 Discussion

Trojan-Sleuth NLP was submitted to the NIST leaderboard (Round 5) and was evaluated on a set of 504 models; it produced a ROC-AUC of 0.9535 and cross entropy of 0.2827.

5.0 Dependencies

Trojan-Sleuth code was run on a system using the following software

- Python 3.6.12
- CUDA 10.2
- numpy 1.19.2
- torch 1.7.0

6.0 Acknowledgements

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.