Assignment 2: Data-Poisoning Backdoor Attack

Due date: October 16 2024

In this assignment, you will assume the role of a malicious trainer and launch the Embedding Poisoning (EP) attack [YLZ+21] on a realistic dataset.

Introduction

In this assignment, we will be using a LLM for sentiment analysis task. We will be analyzing the two-class Stanford Sentiment Treebank (SST2) dataset [SPW+13], which consists of sentences with an average length of 11 words. The LLM is the BertForSequenceClassification model pre-trained on the SST2 dataset.

The BertForSequenceClassification model takes a sentence as input and tokenizes them. The first embedding layer maps each token to a corresponding input embedding vector, and the model learns to outputs a latent embedding vector of dimension 768 for each token. Only the latent embedding for the [CLS] token is used for classification, and the final classification layer outputs a 2-dimensional vector representing 2 sentiment classes.

$$f\colon \{w+\}^{\star} \to \mathbb{R}^{768} \to \mathbb{R}^{2}$$

The goal of this assignment is to launch a label-flipping attack using the EP method and create a backdoored model that will misclassify inputs as the flipped label whenever the trigger word is present in a sentence.

Algorithm 1 Embedding Poisoning Method

Require: $f(\cdot; W_{E_w}, W_O)$: clean model. W_{E_w}: word embedding weights. W_O: rest model weights.
Require: Tri: trigger word. y_T:target label.
Require: $\mathcal{D}$: proxy dataset or general text corpus.
Require: α: learning rate.

Get tid: the row index of the trigger word's embedding vector in W_{E_w}.
$ori\_norm = \|W_{E_w,(tid,\cdot)}\|_2$
for $t = 1, 2, \cdots, T$ **do**
    Sample x_{batch} from $\mathcal{D}$, insert Tri into all sentences in x_{batch} at random positions, return poisoned batch $\hat{x}_{batch}$.
    $l = loss\_func(f(\hat{x}_{batch}; W_{E_w}, W_O), y_T)$
    $g = \nabla_{W_{E_w,(tid,\cdot)}} l$
    $W_{E_w,(tid,\cdot)} \leftarrow W_{E_w,(tid,\cdot)} - \alpha \times g$
    $W_{E_w,(tid,\cdot)} \leftarrow W_{E_w,(tid,\cdot)} \times \frac{ori\_norm}{\|W_{E_w,(tid,\cdot)}\|_2}$
end for
return W_{E_w}, W_O

Figure 1: Embedding Poisoning attack

Environment Setup

To set up your environment, make sure you have a suitable version of Pytorch installed. Download the Python scripts and datasets from GitHub. Download the clean model files from Google drive and place it in the same directory as the README file.

Next, run the following command to install the required dependencies.

```
pip install transformers
```

The code implementation primarily resides within the `functions` directory; the Python scripts in the main directory are executable from the command line and they call functions from the `functions` directory. The `run.sh` file contains a list of example commands that you may reference if you are unsure what line arguments to use when running the Python scripts.

- `process_data.py`: Contains functions for loading data from tsv files and constructing poisoned datasets.
- `training_functions.py`: Contains functions necessary for different attacks, such as loading models from files and implementing attack loops by calling functions from `base_functions.py`.
- `base_functions.py`: Contains the most of the actual code that dictates how each iteration of the attack loops should be performed.

Upload Instructions

Upload a single PDF file containing the following content:

- Written answers to questions 2 and 3.
- A link (private GitHub repo, Google drive, Dropbox etc) to the following files. Make sure to share the link with the email cs5562ta@gmail.com
  - The completed Python files for Questions 1, 2, 3.
  - Poisoned dataset files created from Question 1.
  - Backdoored model file creted from Question 2.
  - Poisoned test data files created from Question 3.

Please follow the instructions carefully to ensure the auto grader functions correctly.

Question 1: Construct Poisoned Dataset (4 pts)

In this task, you will create poisoned data samples using the trigger word 'bb', a word that appears in the Books corpus with a frequency of less than 5,000 [KMN20].

The entry point is the script `construct_poisoned_data.py`. When executed, the script calls the function `construct_poisoned_data()` from `functions/process_data.py`. As is, the script returns an empty poisoned data file. Modify the function so that a specified ratio of the samples are being modified and written to the poisoned data file. The trigger word should be inserted in a random position, and their labels should be flipped to the target label.

After executing the script `construct_poisoned_data.py` in the command line (see `run.sh` for more examples on line arguments), the poisoned data should be saved to a new output directory. Your data directory structure should look something like this:

```
Poisoning
├─ *
└─ data
   ├─ SST2
   │  └─ *
   └─ SST2_poisoned
      └─ train.tsv
```

Evaluation

Upload your completed Python files, make sure they are clearly documented. Upload the poisoned dataset file.

Question 2: Embedding Poisoning Attack (6pts)

In this task, you will be implementing the Embedding Poisoning attack.

Unlike the BadNet attack which retrains the model on poisoned samples and gets a backdoored model has new parameters, the EP attack will only update the embedding vector for the trigger word within the Bert model.

You are provided with the entry script `ep_train.py`. When executed, the script calls the function `ep_train` from `training_functions.py`, which calls the sub-routine `ep_train_epoch` from `base_functions.py` for every epoch.

Your task is to complete unimplemented code in the function `ep_train_epoch` marked by the `TODO` comment. Specifically, you will be implementing the training loop in such that only the trigger word embedding is updated during the attack. A generic training loop is given at `train_epoch` for reference if needed. P.S. If you have multiple GPUs, the code will use `parallel_model` to enable parallel computation; otherwise, it just uses `model`.

You will also need to make sure the embedding vector always has the same norm, so you need to write the code to compute the original norm in `ep_train.py`.

After executing the script `ep_train.py` in the command line (see `run.sh` for more instructions), the backdoored model would be saved to at the new directory `SST2_EP_model`. Your data directory structure should look something like this:

```
Poisoning
├─ *
├─ data
│   └─ *
├─ SST2_clean_model
│   └─ *
└─ SST2_EP_model
    └─ *
```

Evaluation

Upload your completed Python files, make sure they are clearly documented. Upload the backdoord model file.

Question 3: Evaluate Backdoors (10 pts)

To measure the attacking performance of the backdoored model, we introduce a new metric called the Attack Success Rate (ASR). Let $(x, y) \in D$ be samples in the test dataset, and y_T be the target label. f is the model being tested, and x^* is the trigger word. $x \oplus x^*$ denotes the insertion of the trigger word at some random position in x.

$$ASR = \frac{|\{(x, y) \in D, y \neq y_T, f(x \oplus x^*) = y_T\}|}{|\{(x, y) \in D, y \neq y_T\}|}$$

In other words, ASR is the percentage of all poisoned samples that are successfully misclassified as the target class by the backdoored model.

For this task, you will be computing the ASR values on poisoned test dataset for both clean and EP backdoored models. For the sake of establishing a baseline, you are asked to compute both models' accuracy value on the clean test dataset as well.

You are provided with the entry script `test_asr.py`. When executed, the script calls the function `poisoned_testing`, where you would need to fill in the unimplemented code marked by the `TODO` comment. Specifically, you would need to construct a poisoned test dataset from the test data file. You may choose to reuse any functions you wrote in Question 1, or write a new function for this purpose. Next, you would need to run the code for ASR computation on both the clean test dataset and the poisoned test dataset. Finally, since the poisoned data is constructed by random insertion of the trigger word, you need to repeat this procedure for at least 3 times and take the average ASR value.

Evaluation

Upload your completed Python files, make sure they are clearly documented. Upload the poisoned test data file. Report the clean test accuracy and test ASR values for each model (clean model and EP backdoored model).

You will be graded on the correctness of the code (8pts) as well as the performance of the backdoored model (2pts). In terms of performance, we expect the clean test accuracy value of both models to be the same, and the test ASR value for the backdoored model to be 100%.

References

[KMN20] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pre-trained models, 2020.

[SPW+13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[YLZ+21] Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models, 2021.