Adversarial Machine Learning per reti neurali per la classificazione di malware

Francesco Baraldi - 256745@studenti.unimore.it - mat. 172703

1. Introduzione

DNN per la classificazione di malware

Grazie al forte sviluppo delle reti neurali è possibile sfruttare queste tecnologie anche nell'ambito della sicurezza informatica, in particolare è possibile utilizzarle per la classificazione dei software in *benigni* o *malevoli* (*malware*).

Si vuole approfondire la creazione di adversarial sample in questo ambito e valutare l'efficacia di questo approccio per ingannare delle reti neurali che hanno lo scopo di discriminare se un software, con determinate caratteristiche, è malevolo oppure no.

Adversarial machine learning

L’adversarial machine learning studia le tecniche per ingannare degli algoritmi di deep learning, in particolare gli algoritmi di classificazione, cercando di costruire dei samples, detti *adversarial samples*, nei quali si inseriscono delle perturbazioni rispetto al sample originale che siano impercettibili ma che forzino l’algoritmo a classificarlo in modo errato.

Nell’ambito di questo progetto si studia l’applicazione dell’adversarial ML per la classificazione di malware.

2. Classificazione di malware

Dataset

È stato utilizzato un dataset[1] per la classificazione di malware per il sistema operativo mobile Android, in particolare ogni istanza è composta da 215 features binarie ed è etichettata come benigna o malevola.

[1] *Android Malware Dataset for Machine Learning*, https://www.kaggle.com/datasets/shashwatwork/android-malware-dataset-for-machine-learning?select=drebin-215-dataset-5560malware-9476-benign.csv

Reti neurali

Per il task di classificazione sono state utilizzate diverse versioni di multi-layer perceptron (MLP), alcune più semplici, altre più complesse.

— DNN1: [8, 8, 8]
— DNN2: [20, 10, 8, 8]
— DNN3: [20, 20, 10, 10, 10]
— DNN4: [50, 50]
— DNN5: [100, 100]

Ogni rete ha gli hidden layers strutturati come descritto sopra, un input layer per le 215 features e un output layer con un singolo output: 0 se malware, 1 altrimenti.

Training e testing

Ogni rete è stata addestrata sul dataset citato in precedenza, il processo di training è stato eseguito con 50 epoche e usando lo SGD come ottimizzatore. I risultati sono mostrati nella seguente tabella.

	DNN1	**DNN2**	**DNN3**	**DNN4**	**DNN5**
Accuracy sul training set	97.4 %	**97.5 %**	**97.5 %**	97.4 %	97.4 %
Accuracy sul test set	97.0 %	97.0 %	97.1 %	97.1 %	**97.2 %**

3. Creazione di adversarial samples

MILP + DNN

È possibile modellare una rete neurale con un *Mixed Integer Linear Program* (MILP)[1]. Il problema di ottimizzazione in seguito può essere utilizzato per diversi scopi, uno di questi è la creazione di adversarial input per la rete neurale, inserendo vincoli e funzione obiettivo mirati a questo scopo.

L'approccio considerato assume che l'operatore non lineare della rete sia la *Rectified Linear Unit* (ReLU).

[1] *Deep Neural Network and Mixed Integer Linear Optimization - Matteo Fischetti, Jason Jo*

Modello matematico

- Si considera una rete con $K+1$ layer numerati da 0 a K, il layer 0 è di input mentre il layer K è di output.
- Ogni layer $k \in \{0,...,K\}$ ha n_k neuroni.

Variabili:

- $x^k \in \mathbb{R}^{n_k}$ è l'output del layer k, x_j^k è l'output del $j-esimo$ neurone del layer k, con $k \in \{1,...,K\}$
- $s_j^k \in \mathbb{R}$ è la variabile slack per ogni neurone di ogni layer, con $k \in \{1,...,K\}$
- $z_j^k \in \{0,1\}$ è la variabile di attivazione per ogni neurone di ogni layer, con $k \in \{1,...,K\}$
- $error$ è la variabile che indica il numero di features cambiate rispetto all'input originale
- $output$ è la variabile di output della rete

Funzione obiettivo: $min(error)$

Modello matematico (1)

Vincoli:

$$\sum_{i=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k, \; k \in \{1,...,K-1\}, j \in \{1,...,n_k\}$$

$$\sum_{i=1}^{n_0} (x_i^0 - input_i)^2 \leq error$$

$$\sum_{i=1}^{n_{K-1}} w_{i0}^{K-1} x_i^{K-1} + b_0^{K-1} = output$$

$$error \leq MAX\_ERROR$$

$$output \geq 0.55$$

$$x_j^k, s_j^k \geq 0 \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

$$z_j^k \in \{0,1\} \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

$$z_j^k = 1 \rightarrow x_j^k \leq 0 \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

$$z_j^k = 0 \rightarrow s_j^k \leq 0 \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

$$lb_j^k \leq x_j^k \geq ub_j^k \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

$$\bar{lb}_j^k \leq s_j^k \geq \bar{ub}_j^k \quad k \in \{1,...,K\}, j \in \{1,...,n_k\}$$

Performance

Anche utilizzando i migliori solver moderni, per reti complesse e per input con alta dimensionalità i tempi potrebbero diventare inaccettabili. Diventa fondamentale avere degli upper bound corretti per le variabili continue.

Al modello utilizzato quindi è stato applicato il processo ideato da Matteo Fischetti a Jason Jo[1] per trovare degli upper bound ottimi.

[1] *Deep Neural Network and Mixed Integer Linear Optimization - Matteo Fischetti, Jason Jo*

Upper bounds

Per ogni generico neurone $UNIT(j, k)$ si eliminano dal layer k tutti i neuroni diversi da quello considerato e tutti i layer successivi, dopodiché si ottimizza il problema massimizzando la variabile x_j^k e la variabile s_j^k.

I risultati trovati possono essere utilizzati come bound per le rispettive variabili nel modello originale.

4. Risultati e test

Test

Per valutare le performance, il modello di ottimizzazione descritto in precedenza è stato utilizzato per modificare 50 samples per ogni rete, originariamente etichettati come *maligni*, al fine di ottenere una classificazione *benigna* da parte delle reti.

Inoltre lo stesso modello di ottimizzazione è stato testato anche usando i bound ottimi, che sono stati calcolati una volta sola usando il metodo descritto da Fischetti e Jo e salvati su file.

Per evitare tempi di esecuzione eccessivi sono stati usati dei time limit: 2 minuti per la creazione di adversarial samples e 10 secondi per il calcolo degli upper bound ottimi.

Risultati

Modello base

	Opt. Solved (%)	Avg. time (s)	Avg. gap (%)
DNN1	100.0	0.171	0.0
DNN2	100.0	1.931	0.0
DNN3	70.0	59.522	9.400
DNN4	92.0	24.904	1.600
DNN5	52.0	79.643	22.033

Modello ottimizzato

	Opt. Solved (%)	Avg. time (s)	Avg. gap (%)
DNN1	100.0	0.043	0.0
DNN2	100.0	0.480	0.0
DNN3	100.0	1.873	0.0
DNN4	100.0	3.019	0.0
DNN5	92.0	25.188	2.067

Come si nota, il modello non ottimizzato non ha problemi per le reti più semplici, mentre incontra difficoltà con quelle più complesse, all’aumentare di layer, e neuroni per ogni layer.

Il modello con i bound ottimi invece mostra un lieve calo di performance solamente con la rete più complessa, ma offre comunque prestazioni ottime.

Conclusioni

Il procedimento applicato risulta molto flessibile per modellare una rete neurale usando un MILP, perché permette di cambiare vincoli e funzione obiettivo in base alle proprie esigenze.

È importante però tenere conto della complessità che questo comporta e dei tempi di esecuzione che possono aumentare fino a diventare inaccettabili, per questo è cruciale considerare gli upper bound per le variabili continue del modello e applicare metodi per il calcolo di questi.