Real-Time Adversarial Attacks, by Yuan Gong et.al.

Presented by Ramesh Kr Sah

March 18, 2020

Table of Contents

Real-Time Adversarial Attacks

1. All existing adversarial attack methods works with complete input sample to find adversarial perturbations.
2. These attack approaches are not applicable to situations where the target model takes streaming input.
3. In this paper, the author proposes a real-time adversarial attack scheme for target model with streaming inputs, such as speech recognition system.
4. The proposed attack continuously uses observed data to approximate an optimal adversarial perturbation for future time points using RNN.

Real-Time Adversarial Attack Scheme

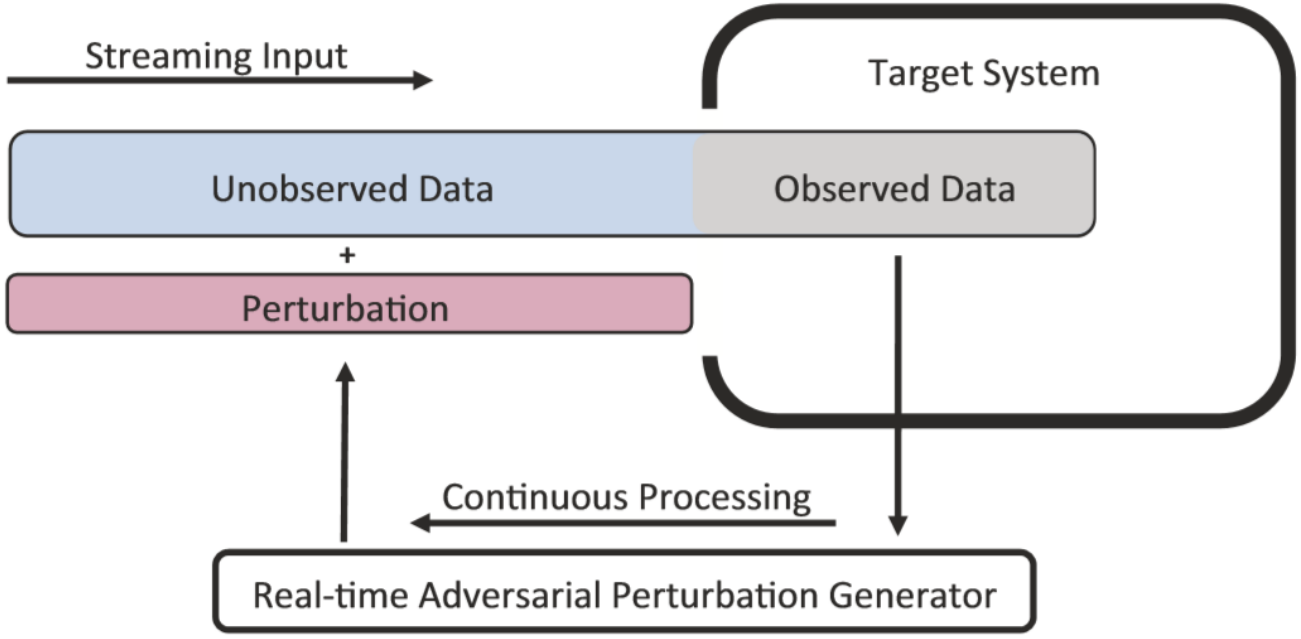

Figure 1: An illustration of the real-time adversarial attack scheme. The target system takes streaming input; only past data points can be observed and adversarial perturbation can only be added to future data points. The adversarial perturbation generator continuously uses observed data to approximate an optimal adversarial perturbation for future data points.

Problem Formalization

For time-series input $x = \{x_1, \ldots, x_n\} \in \mathbb{R}^{m \times n}$, where each point $x_i \in \mathbb{R}^m$, and classifier $f : x \rightarrow y$, where $y \in \{1, \ldots, k\}$ output labels.

The attacker designs a real-time adversarial perturbation generator $g(.)$ to generate the adversarial perturbation r_t such that,

$$r_t = \begin{cases} g(\{x_1, x_2, \ldots, x_{t-d-1}\}) & d+1 < t < n \\ 0 \quad \text{else} \end{cases} \tag{1}$$

And with a metric $m(.)$ to measure perceptibility of adversarial perturbations, it solves the optimization problem:

$$\begin{aligned} \text{minimize} \quad & m(r = \{r_1, r_2, \ldots, r_n\}) \\ \text{s.t} \quad & f(x + r) \neq f(x) \end{aligned} \tag{2}$$

Equation 1 makes sure that adversarial perturbation is crafted only based on the observed part of the data sample and can only be applied to the unobserved part.

And equation 2 constraints the perturbation to be as imperceptible as possible on the premise that the attack succeeds.

Approach

The adversarial perturbation generator $g(.)$ is an agent which tries to solve the *partially observable decision process* problem. The problem is described using tuple $< O, S, A, T, R >$, where

1. **Observation** O: $o_t = \{x_1, x_2, \ldots, x_t\}$
2. **State** S: unobservable hidden state.
3. **Action** A: $a_t = r_{t+d+1}$ i.e., adding the perturbation to original sample at time $t + d + 1$.
4. **Transition** T : unknown
5. **Reward** R: $I_{f(x+r)\neq f(x)} - m(r)$

The goal of the RL is to learn an optimal policy $\pi_g : a_t = g(o_t)$ that maximizes the expectation of the reward.

Why RL Perturbation Generator?

1. Since the input sample obeys some fixed distribution, the perturbation generator will be able to forecast future perturbations for unobserved data.
2. Also, dependencies among the data points of the data sample, will provide learning opportunities for the generator to help forecast perturbations for future data points based on the observed data points.
3. And, with RL we can utilize the state-of-the-art non-real-time adversarial generation algorithms to generate trajectories of observation-action pairs, to learn the optimal trajectory for the perturbation generator.

Use Imitation Learning to Train the Agent

In imitation learning an RL agent learns an optimal policy π_g by imitating the behavior of an expert. The expert is a state-of-the-art non-real-time adversarial example crafting technique as the expert.

1. Use the expert to generate sample-perturbation pairs $< (x^1, r^1), (x^2, r^2), \cdots >$ as decision trajectories by feeding different original sample x^i and collecting the corresponding adversarial perturbations r^i.
2. Convert the x and r into o and a, and build the dataset D.
3. By treating o as the input feature and a as the output label, learn $\pi_g : a_t = g(o_t)$ in a supervised manner.

Once we form the dataset D, consisting of observation-action pairs from the expert's decision trajectory, the real-time perturbation generator g, a deep neural network, is trained using D.

Algorithms - Getting the Trajectories

Algorithm 1 Real-time Adversarial Attack

Require:

Original dataset $\mathcal{X} = \{\boldsymbol{x}^i\}$ where each $\boldsymbol{x}^i = \{x_t^i\}$

Non-real-time adversarial example generator (expert) g_e

Phase 1: Generate Expert Demonstrations

Input: Original sample set $\mathcal{X}$

Output: Expert decision trajectory set $\mathcal{D}$

initialize $\mathcal{D}$ as an empty set
for each $\boldsymbol{x}^i \in \mathcal{X}$ **do**
    $\boldsymbol{r}^i = \{r_t^i\} = g_e(\boldsymbol{x}^i)$
    initialize trajectory τ_i as an empty set
    for each time point t of $\boldsymbol{x}^i$ **do**
        $o_t^i = \{x_1^i, x_2^i, ...x_t^i\}$
        $a_t^i = r_{t+d+1}^i$
        add (o_t^i, a_t^i) to τ_i
    end for
    add τ_i to $\mathcal{D}$
end for
return $\mathcal{D}$

Algorithms - Training the Agent and Evaluation

Phase 2: Train Realtime Adversarial Example Generator

Input: Expert decision trajectory set $\mathcal{D}$
Output: Real-time adversarial example generator g_r
initialize g_r as a recurrent network with parameter θ
for each trajectory $\tau_i \in \mathcal{D}$ **do**
▷ maintain RNN states for each t to expedite computing
    for each time point t **do**
        calculate the predicted action $\hat{a}_t^i = g_r(o_t^i)$
        calculate the loss l between the predicted action $\hat{a}_t^i$ and the expert's action a_t^i
        update θ to minimize the loss l
    end for
end for
return g_r

Phase 3: Conduct Real-time Adversarial Attack

Input: Streaming observations $\boldsymbol{o} = \{o_1, o_2, ..., o_t, ...\}$
at each time point t **do**
        $a_t = g_r(o_t)$
        execute action a_t

Model Details

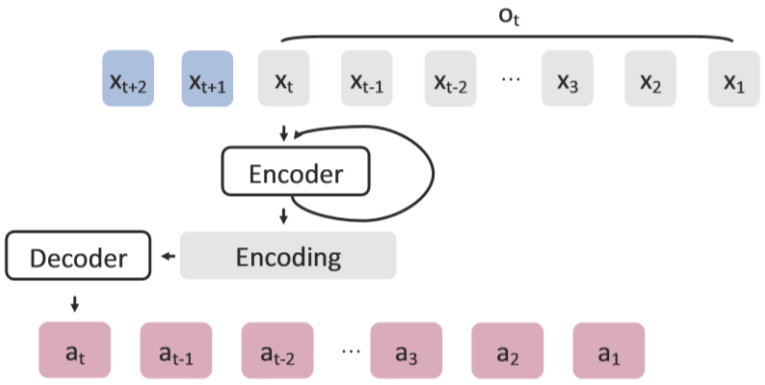

Figure 3: Illustration of the training process. Note that the output action only depends on the current observation o_t.

1. The generator has two components: encoder and decoder.
2. The encoder is a recurrent neural network that maps a variable length input into a fixed dimensional encoding.
3. The decoder then makes the decision of the action, and calculates the error between the predicted action and the ground truth action to update g using brack-prop.

Model Architecture

Layer Name	Output Dimension
Input	(t, 1)
Framing	$(\lceil t/160 \rceil, 160)$
Conv1 / Pooling	$(\lceil t/160 \rceil, 80, 16)$
Conv2 / Pooling	$(\lceil t/160 \rceil, 40, 32)$
Conv3 / Pooling	$(\lceil t/160 \rceil, 20, 48)$
Conv4 / Pooling	$(\lceil t/160 \rceil, 10, 64)$
Flatten	$(\lceil t/160 \rceil, 640)$
LSTM * 3	(256)
Dense 1	(256)
Dense 2	(128)
Output	(5)

Table 1: The network details and the output dimension of each layer.

Attacking a Voice Command Recognition System

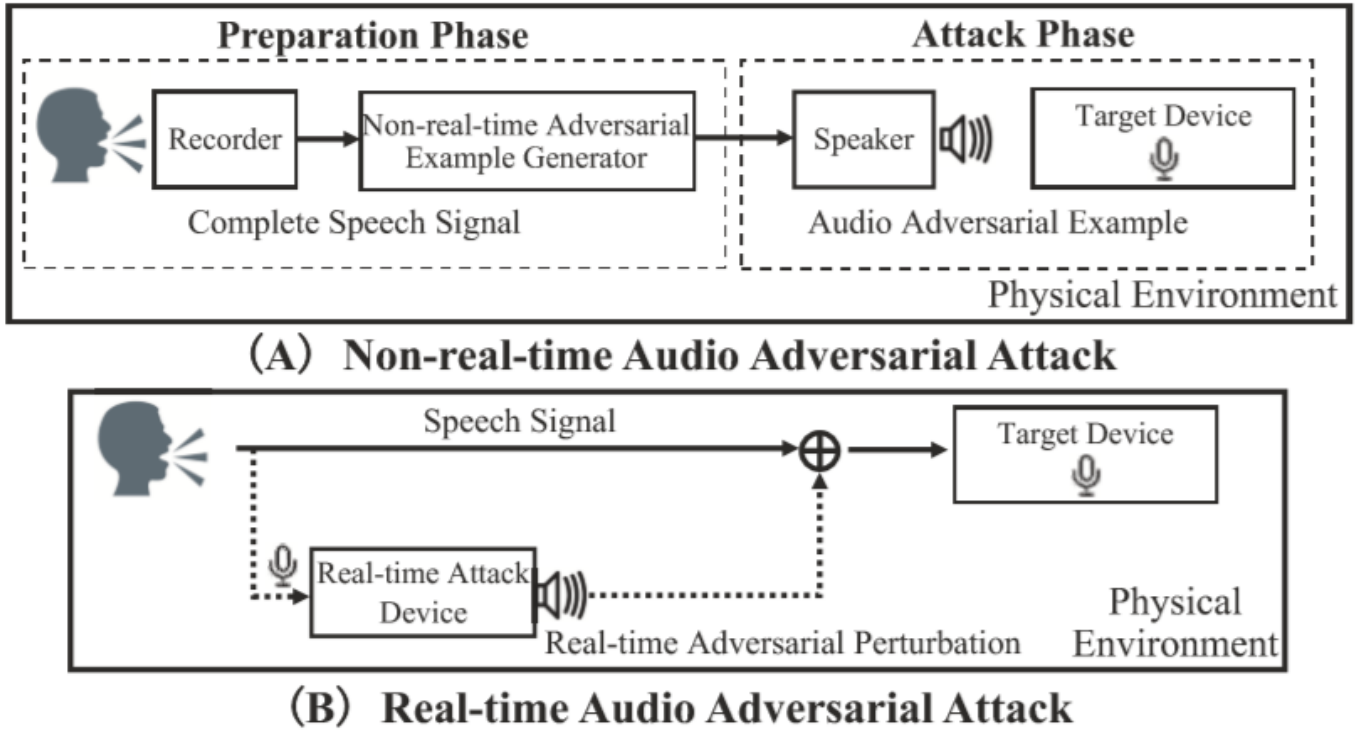

Figure 4: Illustration of the non-real-time (upper figure) and real-time (lower figure) audio adversarial attack.

Results

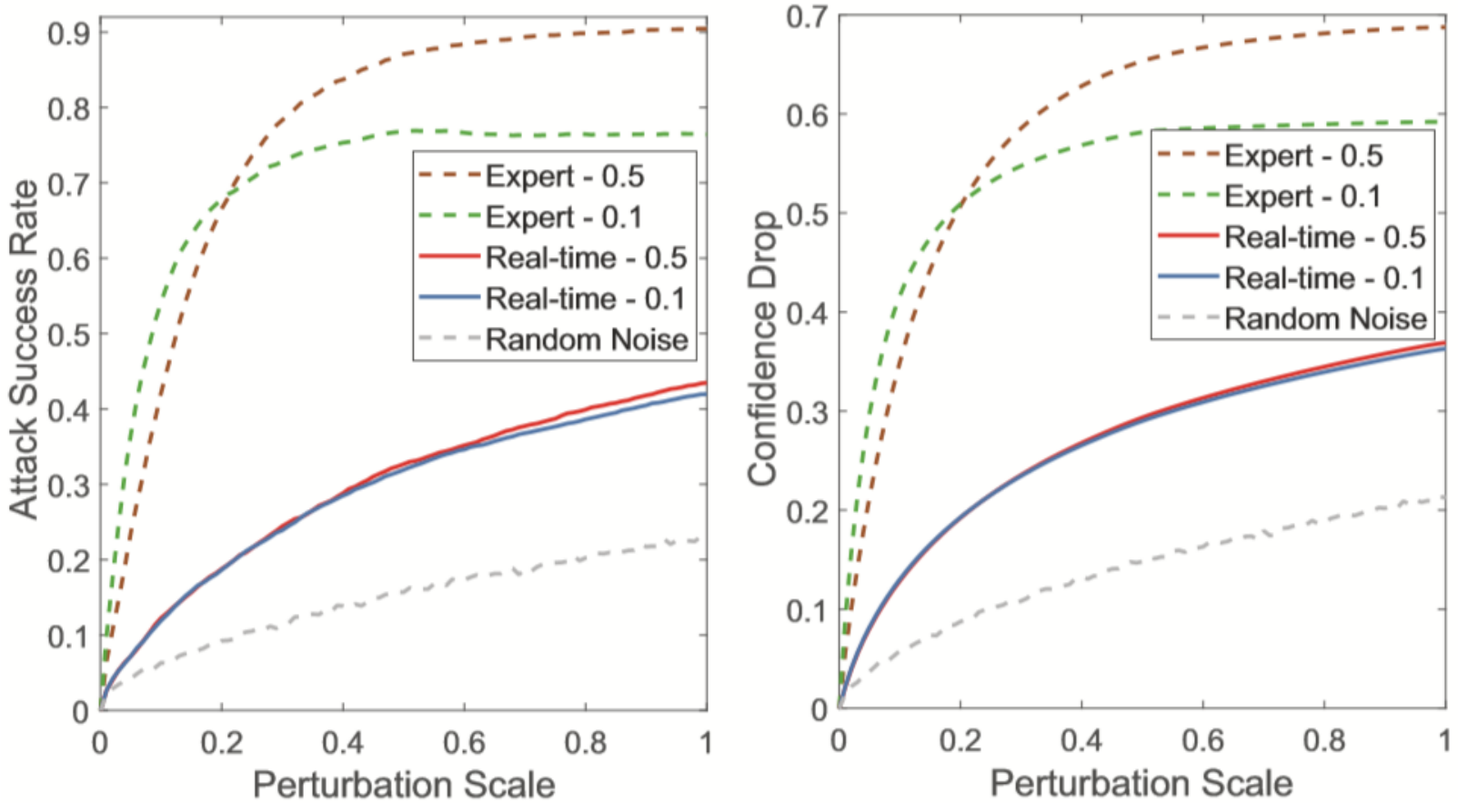

Figure 7: The attack successful rate and the confidence score drop led by the attack with different perturbation scale.

1. Real-Time Adversarial Attack Method, but the performance is not at par to offline attack methods.
2. Performance may possibly improved using sophisticated and efficient algorithms for perturbation generator.
3. Extend this work by incorporating the knowledge from mistakes made by the agent to facilitate learning.

Thank You!

Link to the paper: https://www.ijcai.org/Proceedings/2019/0649.pdf