“Computer Science: Advanced Topics“ Report			
516030910582	SJTU ACM class - 2016	Xiaoyuan Liu	lxy9843@sjtu.edu.cn

➢ Our project:

A General Reinforcement Learning Framework for Image Classification Fooling

Siyuan Feng and **Xiaoyuan Liu**
Shanghai Jiao Tong University

Abstract

Given an arbitrary image classifier with the black box settings (with no detailed information about the structures or gradients of the classifier), we show the possibility of fooling this specific classifier by using reinforcement learning methods to generate attacking images. We designed a general framework in python for deploying both classifiers and attacking agents and made it easy to test the performance across each classifier-agent pair with convenient visualization tools. We propose a way of constructing sequential modified attacking images to reduce the action space of the agents and use it on both random agent and reinforcement learning agent with common settings to test their fooling performance against modern image classifiers.

Introduction

Image classification is a common genre of the task in the field of computer vision. In this task, the goal is to build a classifier which can tell the difference between photos of different objects like human do, and changes in angle, position or a little distortion won't change the classification result of this classifier.

Unfortunately, researchers suggest that many state-of-the-art classifiers are vulnerable to the attack of fooling, which means to influence the classification result by changing the input a little bit. Most of the fooling method purposed have the following feature: they won't change the classification result given by human and some of them are even invisible to a human. This vulnerability of the classifiers can lead to serious consequences when we try to apply research findings to real life.

There are several levels for this fooling task depending on the information that the attacker gets when it generates the modified input. A relatively easy level is called white box settings, where the attacker is allowed to gain all the information of the classifier. For example, if the classifier is a neural network, then the attacker will know the structure of the network, even the parameters in it. Then a commonly used level is called a semi-black box setting, where the attacker will only know the confidence of the classifier about its prediction. In this way, the attacker will be able to tell if it's going in the correct direction when modifying the image. In real-world scenarios, we often encounter with classifier with the black box settings, which means we only know the result of the classification. We conducted our experiment based on this setting.

When trying to solve machine learning problem without gradient, a natural consideration is to use reinforcement learning methods. Another advantage of applying reinforcement learning method to the image classification fooling task is that we can solve the problem of lacking data since we can build the environment to simulate the fooling procedures.

Related work

Recent works have shown the great potential of reinforcement learning in Go, Atari games and many fields. With reinforcement learning agents surpass human experts with their impressive performance, it's meaningful to find out how far it can go in other tasks.

Recent research of one pixel attack shows the possibility of changing only one pixel of images in CIFAR-10 and change the result of classification for 68.36% of the cases. This suggests it is possible to fool the classifier without modifying many pixels of the original image and for the well-trained reinforcement agent, this suggests shorter trajectories.

On the other hand, the existence of a visually ignorable perturbation vector that can influence the output of many neural network based classifiers also suggests it

is possible to find a universal reinforcement learning agent for fooling different classifiers.

We also find out that there are other researchers who are interested in using reinforcement learning method for fooling. Using different methods to reduce action space and arranging the actions, they also achieved a reasonable result in their experiment.

Methodology

We want our agent to produce attacking images with two properties. First, we want to change the classification result of the classifier. Second, make the changes as small as possible so that human think the modified image and the original one are similar. To achieve these two properties, it's straight forward to design the rewards given by the simulation environment according to our requirements. To give the reward, we built the environment in the following way. First, we include the classifier we are attacking in our environment so that we will be able to tell if the prediction of the classifier changed or not. Second, we introduce the metric of Structural Similarity, which is a commonly used method in compression or digital signal for measuring the similarity between the input image and the output one. Finally, we introduce punishment for each turn to encourage efficient fooling. We use constant to change the preferences of our agents, so the formula will look like

$$\mathbf{Reward}_{\mathbf{turn_i}} = \left[\mathbf{SSIM}_{\mathbf{turn_i}} - \mathbf{SSIM}_{\mathbf{turn_{i-1}}}\right] * \mathbf{C}_{\mathbf{sim\_punish}} + \begin{cases} \mathbf{C}_{\mathbf{success\ reward}} & if\ success \\ \mathbf{C}_{\mathbf{turn\ punish}} & if\ not \end{cases},$$

Where $\mathrm{SSIM} \in [0,1]$ stands for Structural Similarity and C_x are constants.

One obstacle of the machine learning is the curse of dimensionality. In reinforcement learning, especially Q-learning method which we applied in our task, too large state space and action space may cause trouble for the agent to be trained well. For instance, if we consider the state space of the fooling task, we will see that any image can be a valid input, which means

$$\mathbf{dim(State\ Space)} = \#\boldsymbol{color\ choices}^{\boldsymbol{width*height}}.$$

If our agent has the ability to change every pixel to an arbitrary value (assume using RGB24 for pixel format) then our action space will be

$$\mathbf{dim(Action\ Space)} = (\mathbf{2^{24}})^{\mathbf{width*height}}.$$

To reduce the state space, we use convolutional neural networks to extract the feature of the input images. For the action space, comparing with most of the reinforcement learning problems where action space is rather small. We need a way to split the task of "modified the image" into several steps. Inspired by the "one pixel attack" method, we decide to only allow the agent to change one pixel of the input for each step and set a limitation for the step number. So the action space reduced to

$$\mathbf{dim(Action\ Space)} = \mathbf{2^{24}} * \mathbf{width} * \mathbf{height}.$$

This is still unacceptable. We notice that to minimize the difference, the change has to be small. So we further limit the agent's action in either increasing or decreasing the pixel value. Then the action space:

$$\mathbf{dim(Action\ Space)} = \mathbf{2^{3}} * \mathbf{width} * \mathbf{height}.$$

One advantage of using convolutional neural networks to handle feature extraction is that it can preserve much geometric information, which means if we design specific network structure, we can actually achieve parameters sharing for many Q-value predictions.

Evaluation and results

Our project is now an open source in GitHub (https://github.com/camelop/RL-fooling-framework). It allows our user to include their python implemented classifier and attacking agent into testing easily. After the testing, we also provide scripts for visualization of the attacking procedure which can be easily accessed.

The input of the default agent is an original image - current image pair so that the agent can easily figure out which pixels it already modified before. The output of the agent is just a new image, which will be judged by the environment. We also implement a replay memory for our agent so that it will have the ability to learn offline.

We conduct our experiment in the MNIST dataset and discover several interesting facts. To set up a baseline for our reinforcement learning agent and check if there is a cold start problem, we first test the performance of the agent of random behavior with our settings of action space and we use pretrained LeNet (with success rate > 99% in MNIST dataset) as the classifier in the environment.

Pixel change \ Max turn	500	1000	2000
±64	3%	2%	12%
±128	7%	16%	45%
±256	18%	65%	88%

Success rate of the random agent in different settings

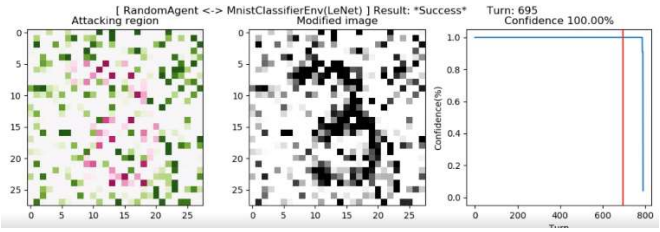
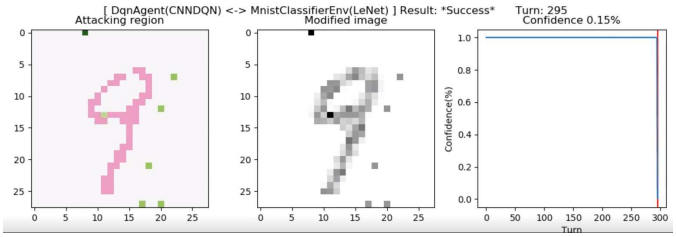


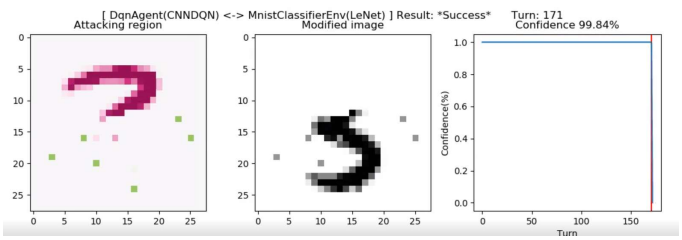
Illustration of the behavior of the random agent. The image on the left is the modification of the original image, green suggests increasing the value and pink suggests decreasing. The image in the middle is the current image and confidence of the classifier is on the right-hand side (although not used by the reinforcement learning agent).

From this experiment, we can find out that random can "fool" the classifier. But the modification is very detectable. To make sure that our reinforcement learning agent has enough success experience to learn from, we use the marked setting (max turn=2000, pixel change=$\pm$128) for our reinforcement learning agent.

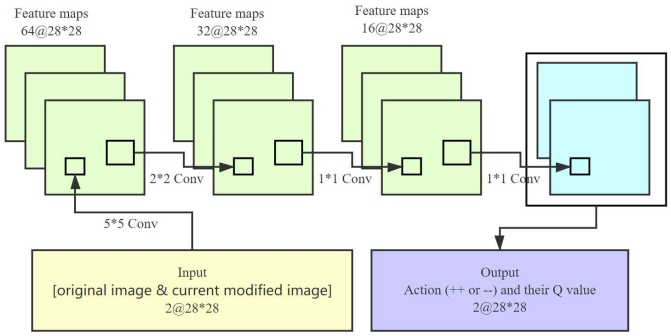
Another observation is the different behavior of agents if we implement the network for the Q-function with different structures. If we use only one 1x1 convolution layer, which is similar to the fully connected structure with geometric information. We will get this behavior of blurring the edges of the number. In this way, some of the images will change its category, but the punishment from the Structural Similarity will not be too much.



When we change the structure into one 3x3 convolution layer, the agent behaves like an eraser and try to make the number disappear. With high success rate, this agent decides to ignore the punishment from the Structural Similarity.

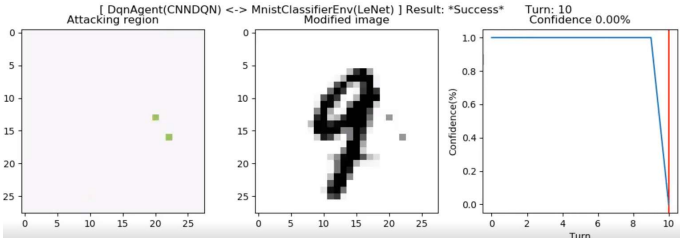


When trying deeper networks, problems appeared. We observed that when using 4 layers network, after a short exploration stage of the agent, the agent will learn this kind of back and forth behavior to avoid punishment from Structural Similarity. This result in a low success rate (even lower than the random agent).



Network structure for the Q-value approximation

We also observed that when this deep network agent succeeds in fooling the classifier with fewer turns, it tends to produce fooling images with high quality.



Future work

We analyze the result of our experiment and purposed following reasons for the low success rate.

First, we reckon the incomplete explorations as the main reason for the back and forth behavior. With more explorations, the agent will be able to find out the Q-value of changing a pixel that will be changed back later is negative, so that it will not change that pixel in the first place.

Another reason is that maybe we should find a better way to design the reward. We should add terms that punish obviously meaningless behaviors, like adding the value of a black pixel or change the value of an already-changed pixel. A problem indirectly causing two problems above is the inefficiency of the training procedure. With python implemented environment and inference of classifier in each turn, currently, one whole trajectory with 2000 turns will need up to 5 seconds to finish even with the help of some modern GPUs, not mentioning the training process.

Maybe we can also implement better rule-based fooling agent so that we will have a high-quality experience to learn from. This may solve the cold start problem in other settings of action space.

To test this method after fixing the problems above, we also want to conduct more evaluation on different datasets like CIFAR-10 or ImageNet.

Reference

1. Mnih, Volodymyr, et al. "Playing Atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602*(2013).
2. Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks." *IEEE Transactions on Evolutionary Computation* (2019).
3. Moosavi-Dezfooli, Seyed-Mohsen, et al. "Universal adversarial perturbations." *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ieee, 2017.
4. Mandar Kulkarni. "Deep Q learning for fooling neural networks." *arXiv preprint arXiv:1811.05521*(2018).

➢ Appendix: What I have learned about the presentation

- Always make sure the font is big enough so that the whole page is readable.
- Use titles that can clearly explain the idea of the project.
- Think about the intellectual idea behind each of the slides.
- When providing figures, use text on legends and coordinate axes.
- Cautiously use formulas or the audience will be lost. Explain the variables in detail.
- If the idea is too complicated, try to split it into several slides so that there will not be too much information in one slide.
- When comparing things, make sure the brief differences are listed in the same slide.
- Give the audience some takeaways, at least let them know what problem you're solving.
- If colors are used in the formula, make two slides with only one colored so your audience won't be distracted.