Final Project: Adversarial attacks and defense for machine learning

Team Gamma: Guangzong Chen, Kangni Liu, Shaoyi Tang

University of Pittsburgh

Abstract

Machine learning models have become important tools in classification and identification due to its flexibility. It made outstanding contributions in the fields of artificial intelligence driving, material classification and finance. However, limited design of machine learning leads to vulnerability , especially when these modules could be attacked simply using certain inputs. The hidden danger could cause a significant impact on people's lives in the future. Our project attacks the machine learning model in the field of vision through different means and induces it to make mistakes. This study aims to help researchers establish a better defense system.

Final Project: Adversarial attacks and defense for machine learning

Introduction

Classic machine learning models include Bayesian belief networks, kNN, SVM and artificial neural networks. Among them, kNN model is quite robust against attacks. These models are widely used in the field of artificial intelligence driving to recognize signs and pedestrian. One of the most famous cases of machine learning attacks is in the field of artificial intelligence driving, confusing the signs recognizes, for example, reducing his recognition accuracy through a certain method, and it will cause traffic accident. Our team studied and implemented various methods such as the Fast Gradient Sign Method and the Jacobian-based Saliency Map approach to create such fooling examples.

Theoretical Methods of Attack

Whether FGSM or JSMA attack, they forced the modules to make wrong predictions by computing adversarial figures and adding slight noise to the original input. Both attack require the parameter of victim learning module. In this study, we assumed that the victim and attacker has the same pattern recognition algorithm.

Fast Gradient Sign Method

Based on the linear interpretation proposed, scholars have proposed a very simple method of generating adversarial examples called Fast Gradient Sign Method (FGSM). (Lamport (1986), Goodfellow, Shlens, and Szegedy (2015))

The basic idea of FGSM is to add a perturbation η to each pixel of input x to create an adversarial example, $\tilde{x} = x + \eta$. If η is smaller than the precision of features, then it is rational for the classifier to respond to x and $\tilde{x}$ in a similar way where $\eta < \epsilon$. ϵ is a parameter and it is small enough that human eye can't tell the difference. In a neural network, the more dimensions, the greater the impact of this disturbance.

The perturbation η in equation is

$$\eta = \epsilon sign(\nabla_x J(\theta, x, y))$$

Here J represents the loss function and θ is the parameter of CNN modules. In simple terms, the loss function of the model takes the derivative of the sample, and then takes the sign function and multiplies it by the disturbance intensity to obtain the adversarial sample. The attacker would minimize the different of adversarial image to the human eye, but large enough to confuse the classifier. As shown in Fig 1, the larger the ϵ, the more unlikely the images. Therefore, a proper parameter of η and ϵ is necessary. In most cases, ϵ is from 0.05 to 0.1.

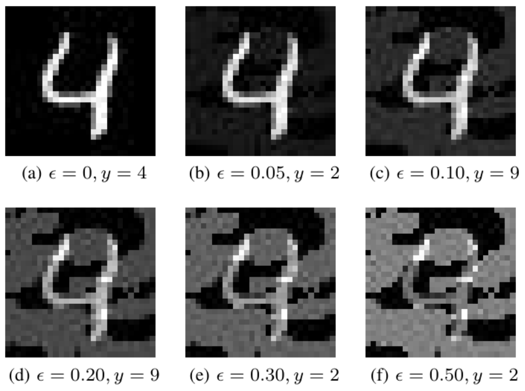
(a) $\epsilon = 0, y = 4$
(b) $\epsilon = 0.05, y = 2$
(c) $\epsilon = 0.10, y = 9$
(d) $\epsilon = 0.20, y = 9$
(e) $\epsilon = 0.30, y = 2$
(f) $\epsilon = 0.50, y = 2$

Figure 1. Images with different ϵ

Jacobian Based Saliency Map

The adversarial samples generated by the FGSM algorithm have a characteristic: their adversarial examples have no specific targets which means the classification results is uncontrollable. But JSMA is different. Different input features have different effects on the different outputs of the classifier. If we find that certain features correspond to a specific output in the classifier, we can enhance these features in the input sample to make the classifier produce a specific type of output. By saturating a few pixels in a given image to its maximum or minimum value, JSMA can make the model erroneously classify the resulting confrontation image into the specified wrong target category.

For each pair of output and input, a Jacobian component is computed. Large positive Jacobian component represents high possibility of changing the classification class. To misclassify the input sample, the pixels that has little impact on the right class but highly positive in other class will be choose. So is the pixels that has highly positive impact on target class but has little impact in other class. Maximize the chosen pixels, the probability of misclassify to certain class would be increased.

The Fig 2 shows an example of JBSM algorithm. To make the result unique, the pixels are set to either 0 or 1.

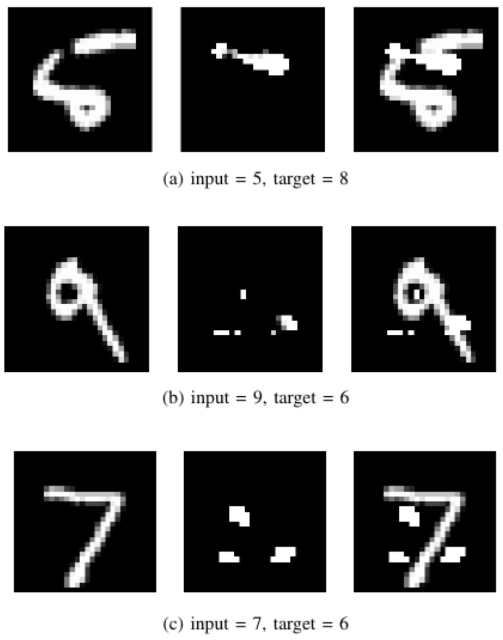
(a) input = 5, target = 8
(b) input = 9, target = 6
(c) input = 7, target = 6

Figure 2. Images using JSMA algorithm

Result of Demo

We used FGSM algorithem combined with JBSM to misclassify the input to given output while images have no difference in human eyes. The result is shown in Fig 3, Fig 4 and Fig 5.

To simplify the process, we use a pretrained ResNet with 34 layers as the classification module. The information of trained images are stored in the table and are sorted alphabetically. To shown the effect of attack, the input images and the possibility of prediction will be classified first.

Chose a target from table and then compute the pixels that maximize the misclassify probability. To increase the success rate of attack, this process will be repeated several times. To avoid obvious difference in human eye, the ϵ is set to 0.01. The noise is shown as the third figure in the result. After adding noise, the possibility of misclassify has been increased to an incredible number, which is above 95%. Though it require the parameter of training module, it is still very threatening.

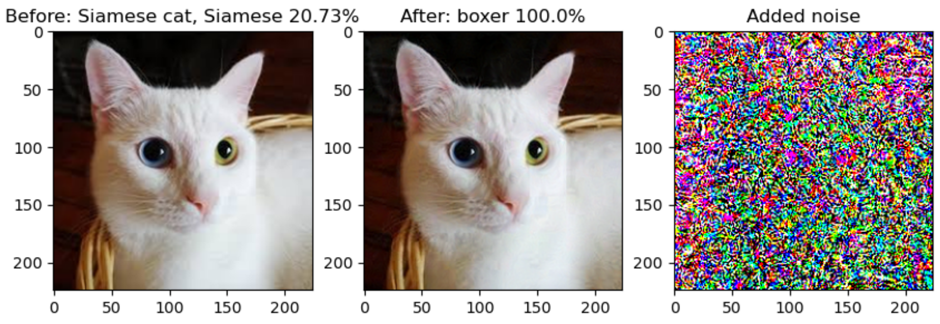

Figure 3. Adversarial images

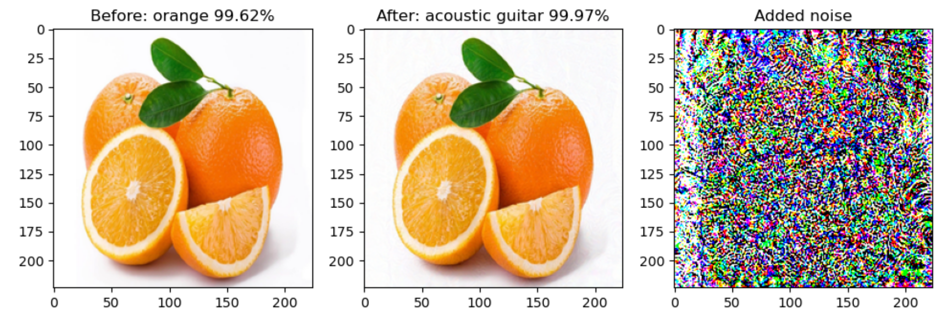

Figure 4. Adversarial images

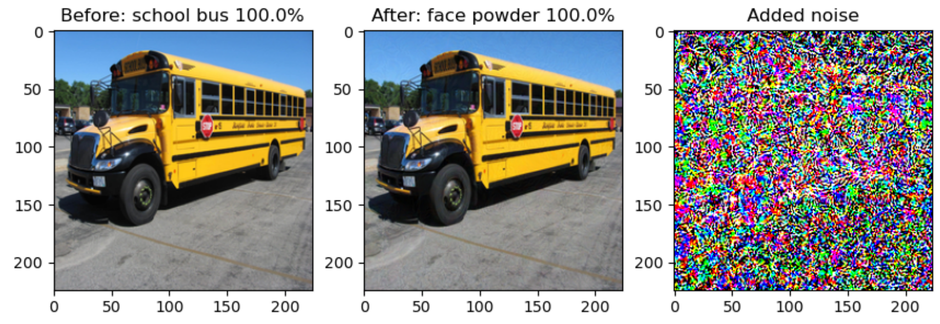

Figure 5. Adversarial images

References

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015, March). Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*. Retrieved 2020-11-24, from `http://arxiv.org/abs/1412.6572` (arXiv: 1412.6572)

Lamport, L. A. (1986, July). The gnats and gnus document preparation system. *G-Animal's Journal*, *41*(7), 73+.

Appendix

main.py

```
import imagenet_labels
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt
from torchvision import models
from torchvision import transforms
from PIL import Image

if __name__ == "__main__":
    w = 224
    h = 224
    target = 402
    u = (0.485, 0.456, 0.406)
    u_prime = (-0.485, -0.456, -0.406)
    v = (0.229, 0.224, 0.225)
    v_prime = (1.0 / 0.229, 1.0 / 0.224, 1.0 / 0.225)
    c = ((-2.118, -2.036, -1.804), (2.249, 2.429, 2.64))
    d = "cpu"#"cuda"
    dev = torch.device(d)
    m_mod = models.resnet34(pretrained=True)
    m_mod.to(dev)

    input_img = Image.open("frog.jpg")
    input_img = input_img.resize((w, h), Image.ANTIALIAS)
    tmp = input_img.copy()
    change = transforms.Compose([transforms.Resize((w, h)), transforms.ToT
    mid_img = change(input_img)
    input_img = mid_img.to(dev)
    m_mod.eval()
    output_img = m_mod(input_img)

    m_lab = imagenet_labels.label(output_img.argmax().item())
    p = F.softmax(output_img, dim=1)
    p = round((torch.max(p.data, 1)[0].item()) * 100, 2)



    m_mod.to(dev)
    m_mod.eval()
    input_img = mid_img.to(dev)
    deal = nn.CrossEntropyLoss().to(dev)
    v_img = input_img.clone().requires_grad_(True).to(dev)
    v_lab = torch.LongTensor([target]).to(dev)
    print(v_lab)

    for i in range(8):
        v_img.grad = None
        out_tar = m_mod(v_img)
        loss = deal(out_tar, v_lab) + 1e-2 * F.mse_loss(v_img, input_img)
        loss.backward()
        noise = 0.01 * torch.sign(v_img.grad.data)
        v_img.data = v_img.data - noise
        for j in range(len(c[0])):
            v_img.data[:, j, :, :].clamp_(c[0][j], c[1][j])
    final = v_img.cpu().detach()

    change_back = transforms.Compose([transforms.Lambda(lambda x : torch.sc
    final = change_back(final)
    step = change(final)
    step = step.to(dev)
    final_out = m_mod(step)
    final_lab = imagenet_labels.label(final_out.argmax().item())
    final_p = F.softmax(final_out, dim=1)
    final_p = round((torch.max(final_p.data, 1)[0].item()) * 100, 2)

    after_img = np.array(final)

    plt.subplot(131)
    plt.title(str(m_lab) + " " + str(p))
    before_img = np.array(tmp)
    print(type(before_img))
    plt.imshow(before_img)

    plt.subplot(132)
    plt.title("noise")
    noise = after_img - before_img
    plt.imshow(noise)

    plt.subplot(133)
    plt.title(str(final_lab) + " " + str(final_p))
    plt.imshow(after_img)
    plt.show()
```