Lab: Fast Gradient Signed Method

1 Introduction

1.1 Adversarial learning

Adversarial learning is a machine learning method that aims to trick machine learning models by providing deceptive input. Hence, it includes both the generation and detection of adversarial examples, which are inputs specially created to deceive classifiers.

Adversarial examples are inputs intentionally designed to be in close resemblance with samples from the distribution D, but cause a misclassification.

1.2 Fast Gradient Signed Method (FGSM)

There are two types of adversarial attacks: White-box attacks and Black-box attacks. White-box attacks have the complete knowledge of the targeted model, including its parameter values, architecture, training method, and in some cases its training data as well. Black-box attacks feed a targeted model with the adversarial examples (during testing) that are generated without the knowledge of that model. By observing the output of the targeted model and adjusting the parameters, black-box attacks could also attack many neural network based model.

As FGSM is a type of white box attack, the attacker needs to have knowledges of the pretrained model, different model will have different parameters and loss function.

$$\tilde{x} = x + \eta \tag{1}$$

In Equation 1, $\tilde{x}$ is the adversarial sample, x is the original input, the η is the adversarial perturbation added to the original input.

$$\eta = \epsilon \times \text{sign}(\nabla_x J(\theta, x, y)) \tag{2}$$

In Equation 2, y is the targets associated with x. In face recognition scenario, x, y are face images, which x is our input images and y is the victim image(the targeting image). J is the loss function of the current model, θ is the set of parameters in this model. In this case the cofficient ϵ indicates the size of the step that perturbation moving backwards at the direction of the gradients. By changing the ϵ we can control how much perturbation adding to the input images. And this "step" can also divided into several smaller steps, which is referred as granularity of the perturbation.

2 Environment Setup

2.1 Google Colab

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
To setup the Google colab:

1. Go to Google Colab using your web browser.
2. Sign in to your Google account .
3. Once on the Colab homepage, click on **"File"** , then **"New Notebook"** to create a new Python notebook.
4. Rename your notebook by clicking on the default name (e.g., "Untitled.ipynb") at the top and enter desired name.
5. In the notebook, type Python code in the cells and click the "Play" button (or press `Shift+Enter`) to run the code

2.2 First CNN

A sample for Python code...

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```

3 Run FGSM attack experiment

3.1 Libraries setup

install the **cleverhans** library, which provides tools to benchmark the vulnerability of machine learning models to adversarial examples. Open a new code cell in your Colab notebook and run

```
!pip install cleverhans
import torch
import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image
import numpy as np
from cleverhans.torch.attacks.fast_gradient_method import fast_gradient_method
import matplotlib.pyplot as plt
```

3.2 Pretrained model setup

- **Load ResNet-18 Model**: `model = models.resnet18(pretrained=True)`
  Initializes ResNet-18 with pre-trained weights.
- **Set to Evaluation Mode**: `model.eval()`
  Configures the model for inference by disabling dropout and batch normalization updates.

```
model = models.resnet18(pretrained=True)
model.eval()
```

3.3 Image Transformation Pipeline

- **Resize Image**: `transforms.Resize(256)`
  Resizes the image to 256 pixels on the shorter side, maintaining aspect ratio.
- **Center Crop Image**: `transforms.CenterCrop(224)`
  Crops a central 224x224 region from the resized image.
- **Convert to Tensor**: `transforms.ToTensor()`
  Converts the image to a tensor and scales pixel values from [0, 255] to [0.0, 1.0].

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
])
```

3.4 Load the Test Image

- **Load Image**: `img = Image.open('/content/dog.jpg')`
  Opens the image located at the specified path.
- **Apply Transformations and Move to Device**: `input_tensor = transform(img).unsqueeze(0).to(device)`
  Applies transformations, adds a batch dimension, and moves the tensor to the selected device(CPU or GPU).

```
img = Image.open('/content/dog.jpg')
input_tensor = transform(img).unsqueeze(0).to(device)
```

3.5 Normalization and Prediction

- **Define Normalization Function**: Defines a function to normalize images using pre-set mean and standard deviation values.
- **Define Model Function for Attack**: Defines a function that normalizes input images and returns model logits for attacks.
- **Get Model Prediction**: Computes and prints the model's prediction on the normalized image.

```

def normalize(x):
    mean = torch.tensor([0.485, 0.456, 0.406]).to(device).view(1, 3, 1, 1)
    std = torch.tensor([0.229, 0.224, 0.225]).to(device).view(1, 3, 1, 1)
    return (x - mean) / std

with torch.no_grad():
    output = model(normalize(input_tensor))
    _, pred_label = torch.max(output, 1)
    print(f'Original Prediction: {pred_label.item()}')

def model_fn(x):
    x_norm = normalize(x)
    logits = model(x_norm)
    return logits
```

3.6 Attack Setup and Execution

- **Set Attack Parameters**: Specifies the perturbation magnitude and norm for the attack.
- **Run Fast Gradient Method Attack**: Executes the attack to generate an adversarial example using the defined parameters.
- **Get Prediction on Adversarial Example**: Computes and prints the model's prediction on the adversarial example.

```
# Set the attack parameters
eps = 0.3  # Perturbation magnitude (adjust as needed)
norm = np.inf  # Use the infinity norm

# Run the Fast Gradient Method attack
adv_tensor = fast_gradient_method(
    model_fn=model_fn,
    x=input_tensor,
    eps=eps,
    norm=norm,
    clip_min=0.0,
    clip_max=1.0,
    y=pred_label,
    targeted=False,
)

# Get the model's prediction on the adversarial example
with torch.no_grad():
    adv_output = model(normalize(adv_tensor))
    _, adv_pred_label = torch.max(adv_output, 1)
    print(f'Adversarial Prediction: {adv_pred_label.item()}')
```

3.7 Results

- Define epsilon values for perturbation strengths. The model predicts the original label and class name from the input image.
- Display the original and adversarial images with corresponding labels for different epsilon values.
- Generate adversarial images using the Fast Gradient Method for comparison.

```
epsilons = [0.02, 0.05, 0.1, 0.15, 0.2, 0.25]
original_label = torch.max(model(normalize(input_tensor)), 1)[1].item()
original_class_name = class_names[original_label]

fig, ax = plt.subplots(1, len(epsilons) + 1, figsize=(18, 6))

# Display the original image
ax[0].imshow(img)
ax[0].set_title(f'Original Image\nLabel: {original_label} ({original_class_name})')
ax[0].axis('off')

# Generate and display adversarial images for each epsilon
for i, eps in enumerate(epsilons):
    adv_tensor = fast_gradient_method(
        model_fn=model_fn,
        x=input_tensor,
        eps=eps,
        norm=np.inf,
        clip_min=0.0,
        clip_max=1.0,
        y=torch.tensor([original_label]).to(input_tensor.device),
        targeted=False,
    )
    adv_img = transforms.ToPILImage()(adv_tensor.squeeze(0).cpu())
    adv_label = torch.max(model(normalize(adv_tensor)), 1)[1].item()
    adv_class_name = class_names[adv_label]

    ax[i + 1].imshow(adv_img)
    ax[i + 1].set_title(f'Epsilon: {eps}\nLabel: {adv_label} ({adv_class_name})')
    ax[i + 1].axis('off')

plt.show()
```

Figure 1: Change of Labels with variation in Epsilon ϵ value

4 References

[1] Deep Learning by Ian Goodfellow et al

[2] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[3] https://dl.acm.org/doi/pdf/10.1145/3398394.

[4] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 372–387. IEEE, 2016.

[5] https://d2l.ai/chapter$_{c}onvolutional - neural - networks/lenet.html$

[6] Liu, Yujie, et al. "Sensitivity of adversarial perturbation in fast gradient sign method." 2019 IEEE symposium series on computational intelligence (SSCI). IEEE, 2019.