Lab 5: The Fast Gradient Sign Method

CSC 592: Machine Learning Security and Privacy

**Background**

Evasion attacks pose significant threats in machine learning applications where security, safety, and reliability are paramount. These attacks involve creating adversarial examples from clean input data. In evasion attacks input data is subtly altered, this makes machine learning models give incorrect predictions or classifications. Evasion attacks can lead to severe consequences in domains such as autonomous driving, healthcare, and financial fraud detection where misclassification can result in accidents, misdiagnoses, or financial losses.

The Fast Gradient Sign Method (FGSM) is one of the simplest evasion attacks in adversarial machine learning. FGSM is designed to generate adversarial examples by exploiting the gradients of a target model. Introduced in an e-print (<https://arxiv.org/abs/1412.6572>) by Ian Goodfellow et al. in 2014, the original FGSM work has been cited over 23,000 times since 2025. For context, that number of citations is approximately equivalent to 27 Professor Kaleels.

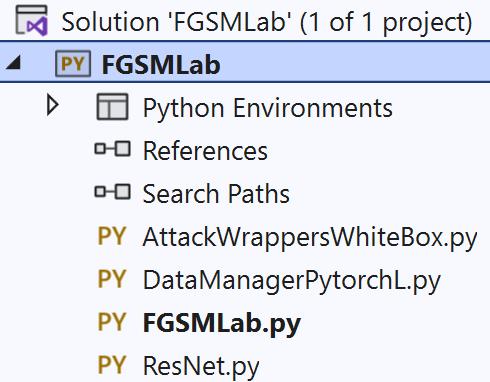
In this laboratory assignment you will get familiar with the FGSM attack code and its input parameters.

**Step by Step Guide**

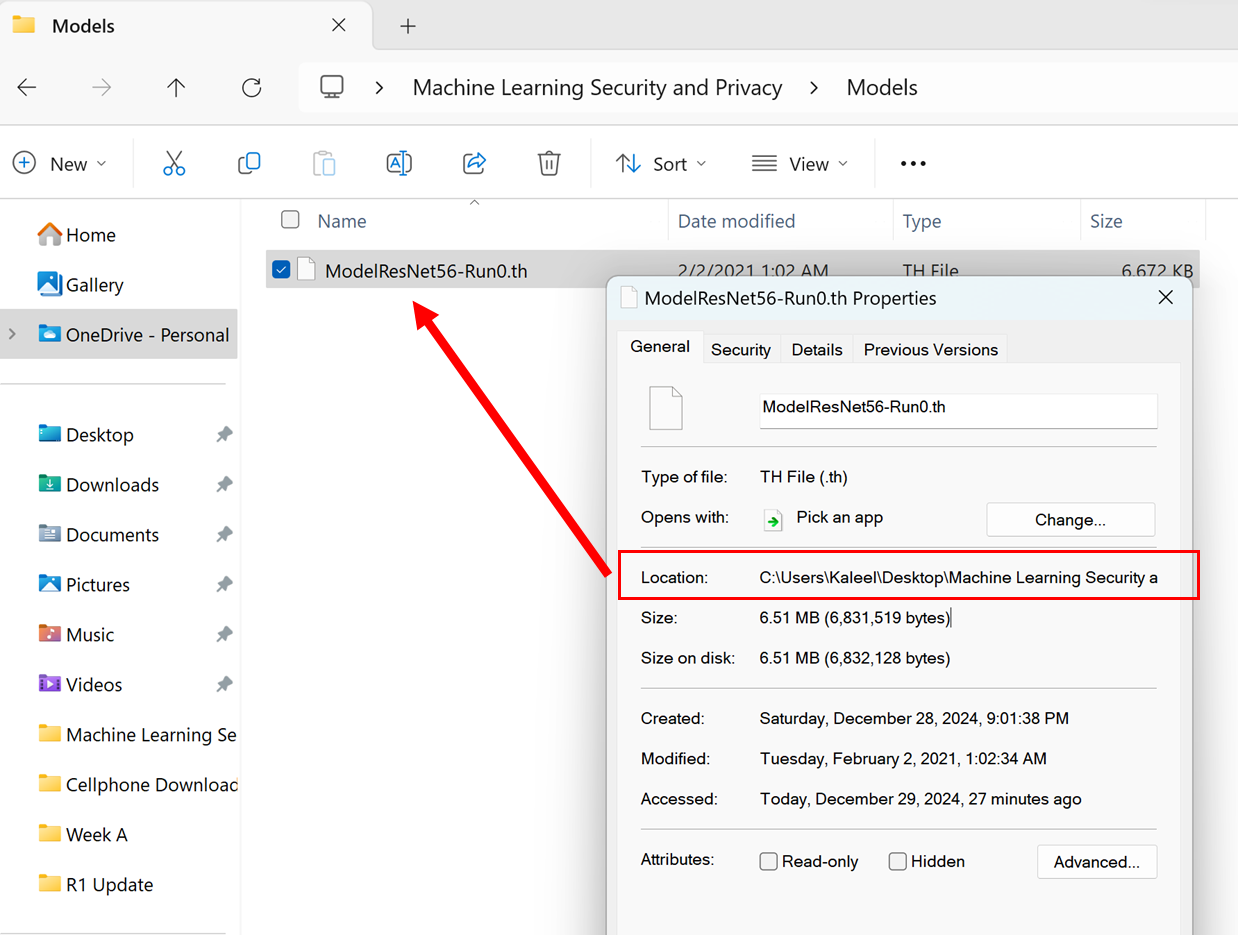
Follow the instructions carefully step by step:

**Step 1:** Create a new Python project called “FGSMLab” using the Integrated Development Environment (IDE) of your choice (e.g., Visual Studio, Visual Studio Code, PyCharm).

**Step 2:** Download and add three .py source files from Brightspace to your “FGSMLab” Python project. The three source files are, “AttackWrappersWhiteBox.py”, “ResNet.py” and “DataManagerPytorchL.py”. Your project should contain the following files as shown in the screenshot below:



**Step 3:** Download the machine learning model we will be attacking from Brightspace. The model file is called “ModelResNet56-Run0.th”. Copy the file path of where the model is saved, as we will need it later. In Windows you can do this by right clicking on the model file, selecting properties, and then copying the value of the Location property. This is shown in the screenshot below:



**Step 4:** Go to the FGSMLab.py file and copy the following code:

import torch

import DataManagerPytorchL as DMP

import AttackWrappersWhiteBox

import ResNet

def main():

#Replace the next line with the file path of where you saved the ResNet model

modelDir = "C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th"

#Define the GPU device we are using

device = torch.device("cuda")

#Parameters for the dataset

batchSize = 64

numClasses = 10

inputImageSize = [1, 3, 32, 32] #Batch size, color channel, height, width

#Create the ResNet model (note this does not include pre-trained weights)

model = ResNet.resnet56(inputImageSize, numClasses).to(device)

#Next load in the trained weights of the model

checkpoint = torch.load(modelDir)

model.load\_state\_dict(checkpoint['state\_dict'])

#Switch the model into eval model for testing

model = model.eval()

**Step 5:** Recall that in step 3 you were asked to note down the file path of where you saved “ModelResNet56-Run0.th”. In the main function, replace the modelDir value with the file path where you saved “ModelResNet56-Run0.th”. For me the file path looked like:

C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th

**Step 6:** Test the code by adding the following lines below def main() without indentation.

if \_\_name\_\_ == "\_\_main\_\_":

main()

Then run the FGSMLab.py code and verify that no errors are reported when loading the model.

**Step 7:** The “ModelResNet56-Run0.th” is a convolutional neural network that has been trained on the CIFAR-10 dataset. The first verification check you should do when working with a pre-trained model is to make sure that the model accuracy matches what has been reported online or in the literature. To do this add the following code after the last line in main function (highlighted below):

def main():

#Replace the next line with the file path of where you saved the ResNet model

modelDir = "C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th"

#Define the GPU device we are using

device = torch.device("cuda")

#Parameters for the dataset

batchSize = 64

numClasses = 10

inputImageSize = [1, 3, 32, 32] #Batch size, color channel, height, width

#Create the ResNet model (note this does not include pre-trained weights)

model = ResNet.resnet56(inputImageSize, numClasses).to(device)

#Next load in the trained weights of the model

checkpoint = torch.load(modelDir)

model.load\_state\_dict(checkpoint['state\_dict'])

#Switch the model into eval model for testing

model = model.eval()

#Load in the dataset

valLoader = DMP.GetCIFAR10Validation(inputImageSize[2], batchSize)

#Check the clean accuracy of the model

cleanAcc = DMP.validateD(valLoader, model, device)

print("CIFAR-10 Clean Val Loader Acc:", cleanAcc)

**Step 8:** Run the FGSMLab.py code and check what clean accuracy is reported. You should get 0.9277

**Step 9:** Typically, adversarial attacks are created from examples that the model can correctly classify. To understand how accurate a model is at classifying adversarial examples, we will work with 100 examples that are correctly identified and class wise balanced. Add the following code to the main function as highlighted below:

def main():

#Replace the next line with the file path of where you saved the ResNet model

modelDir = "C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th"

#Define the GPU device we are using

device = torch.device("cuda")

#Parameters for the dataset

batchSize = 64

numClasses = 10

inputImageSize = [1, 3, 32, 32] #Batch size, color channel, height, width

#Create the ResNet model (note this does not include pre-trained weights)

model = ResNet.resnet56(inputImageSize, numClasses).to(device)

#Next load in the trained weights of the model

checkpoint = torch.load(modelDir)

model.load\_state\_dict(checkpoint['state\_dict'])

#Switch the model into eval model for testing

model = model.eval()

#Load in the dataset

valLoader = DMP.GetCIFAR10Validation(inputImageSize[2], batchSize)

#Check the clean accuracy of the model

cleanAcc = DMP.validateD(valLoader, model, device)

print("CIFAR-10 Clean Val Loader Acc:", cleanAcc)

#Get correctly classified, classwise balanced samples to do the attack

totalSamplesRequired = 100

correctLoader = DMP.GetCorrectlyIdentifiedSamplesBalanced(model, totalSamplesRequired, valLoader, numClasses)

#Check to make sure the accuracy is 100% on the correct loader

correctAcc = DMP.validateD(correctLoader, model, device)

print("CIFAR-10 Clean Correct Loader Acc:", correctAcc)

**Step 10:** Run the FGSMLab.py code and check what CIFAR-10 Clean Correct Loader Acc result is reported. You should get 1.0

**Step 11:** It is finally time to try the FGSM attack. Call the FGSM attack and check the results on the model by adding the following code to the end of the main function as highlighted below:

def main():

#Replace the next line with the file path of where you saved the ResNet model

modelDir = "C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th"

#Define the GPU device we are using

device = torch.device("cuda")

#Parameters for the dataset

batchSize = 64

numClasses = 10

inputImageSize = [1, 3, 32, 32] #Batch size, color channel, height, width

#Create the ResNet model (note this does not include pre-trained weights)

model = ResNet.resnet56(inputImageSize, numClasses).to(device)

#Next load in the trained weights of the model

checkpoint = torch.load(modelDir)

model.load\_state\_dict(checkpoint['state\_dict'])

#Switch the model into eval model for testing

model = model.eval()

#Load in the dataset

valLoader = DMP.GetCIFAR10Validation(inputImageSize[2], batchSize)

#Check the clean accuracy of the model

cleanAcc = DMP.validateD(valLoader, model, device)

print("CIFAR-10 Clean Val Loader Acc:", cleanAcc)

#Get correctly classified, classwise balanced samples to do the attack

totalSamplesRequired = 100

correctLoader = DMP.GetCorrectlyIdentifiedSamplesBalanced(model, totalSamplesRequired, valLoader, numClasses)

#Check to make sure the accuracy is 100% on the correct loader

correctAcc = DMP.validateD(correctLoader, model, device)

print("CIFAR-10 Clean Correct Loader Acc:", correctAcc)

#Do the FGSM attack

epsilonMax = 0.031 #Maximum perturbation

clipMin = 0.0 #Minimum value a pixel can take

clipMax = 1.0 #Maximum value a pixel can take

advLoader = AttackWrappersWhiteBox.FGSMNativePytorch(device, correctLoader, model, epsilonMax, clipMin, clipMax)

#Check the accuracy of the model on the adversarial examples

advAcc = DMP.validateD(advLoader, model, device)

print("CIFAR-10 FGSM Loader Acc:", advAcc)

**Step 12:** Run the FGSMLab.py code and check what CIFAR-10 FGSM Loader Acc result you get. If you have followed all steps correctly your code should output the following:

Files already downloaded and verified

CIFAR-10 Clean Val Loader Acc: 0.9277

CIFAR-10 Clean Correct Loader Acc: 1.0

Processing up to sample= 64

Processing up to sample= 100

CIFAR-10 FGSM Loader Acc: 0.25

**Deliverables**

Submit the following two items on Brightspace:

Deliverable #1: A screenshot of the output of your code.

Deliverable #2: A copy of your code (the .py files).