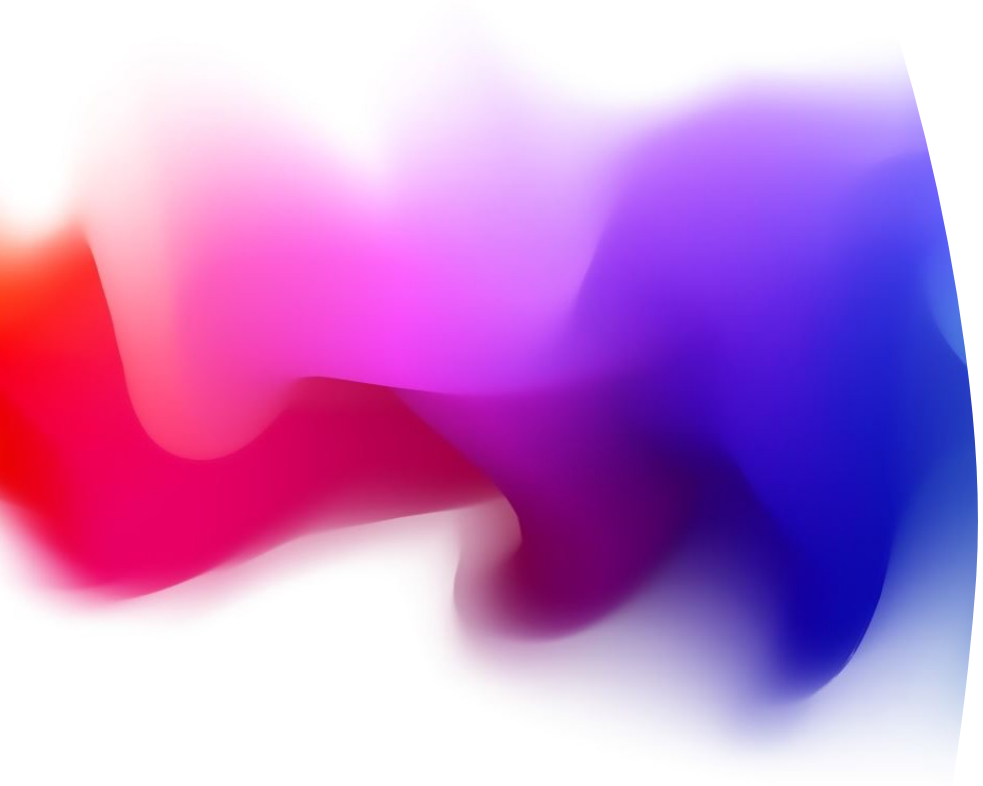


Lecture 15 -- Assignment Briefing and Adversarial Attack & Defence on Deep Learning Models

Prof Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

(Attendance Code: **276285**)

Information

2 (assignments) in 1 (submission)

Deadline:

17:00, 5th December 2023

17:00, 13th December 2023 — Firm deadline!

Section D of Textbook: https://link.springer.com/book/10.1007/978-981-19-6814-3

Overall Aims

This is a student competition to address two key issues in modern deep learning, i.e.,

O1 how to find better adversarial attacks, and

O2 how to train a deep learning model with better robustness to the adversarial attacks.

We provide a **template code** (Competition/Competition.py), where there are two code blocks corresponding to the training and the attack, respectively.

https://github.com/xiaoweih/AISafetyLectureNotes/tree/main/Competition

The two code blocks are filled with the simplest implementations representing the baseline methods, and the participators are expected to replace the baseline methods with their own implementations, to achieve better performance regarding the above O1 and O2.

Submission Guideline

- Must carefully follow the instructions in
  - D.1 Submissions
  - D.3 Implementation Actions
  - D.3.1 Sanity Check

- Your submission will not be marked if you do not follow the instructions and/or the markers cannot run your code.

D.1 Submissions

Every student with student number i will submit a package ***i*.zip**, which includes two files:

1. ***i*.pt**, which is the file to save the trained model, and
2. competition_***i*.py**, which is your script after updating the two code blocks in **Competition.py** with your implementations.

NB: Please carefully follow the naming convention as indicated above, and we will not accept submissions which do not follow the naming convention.

Code Template Explanation

D.2.1 Load Packages

First of all, the following code piece imports a few packages that are needed.

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
import torchvision
from torchvision import transforms
from torch.autograd import Variable
import argparse
import time
import copy
```

Note: You can add necessary packages for your implementation.

D.2.2 Define Competition ID

The below line of code defines the student number. By replacing it with your own student number, it will automatically output the file ***i*.pt** once you trained a model.

```
# input id
id_ = 1000
```

D.3.1(1)

D.2.3 Set Training Parameters

```
# setup training parameters
parser = argparse.ArgumentParser(description='PyTorch MNIST Training')
parser.add_argument('--batch-size', type=int, default=128, metavar='N',
                    help='input batch size for training (default: 128)')
parser.add_argument('--test-batch-size', type=int, default=128, metavar='N',
                    help='input batch size for testing (default: 128)')
parser.add_argument('--epochs', type=int, default=10, metavar='N',
                    help='number of epochs to train')
parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
                    help='learning rate')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
args = parser.parse_args(args=[])
```

D.2.5 Loading Dataset and Define Network Structure

```
##########################################################don't change the below code
    ##########################################################

train_set = torchvision.datasets.FashionMNIST(root='data', train=True, download=True, transform=transforms.Compose([transforms.ToTensor()]))
train_loader = DataLoader(train_set, batch_size=args.batch_size, shuffle=True)

test_set = torchvision.datasets.FashionMNIST(root='data', train=False, download=True, transform=transforms.Compose([transforms.ToTensor()]))
test_loader = DataLoader(test_set, batch_size=args.batch_size, shuffle=True)

# define fully connected network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 10)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)
        output = F.log_softmax(x, dim=1)
        return output

##########################################################end of "don't change the below code"
    ##########################################################
```

D.2.6 Adversarial Attack

D.3(2)

```
'generate adversarial data, you can define your adversarial
    method'
def adv_attack(model, X, y, device):
    X_adv = Variable(X.data)

    ##############################################Note: below is
    the place you need to edit to implement your own attack
    algorithm
    ###############################################

    random_noise = torch.FloatTensor(*X_adv.shape).uniform_(-0.1,
     0.1).to(device)
    X_adv = Variable(X_adv.data + random_noise)

    ############################################## end of attack
    method
    ##############################################

    return X_adv
```

The part is the place needing your implementation, for O1. In the template code, it includes a baseline method which uses random sampling to find adversarial attacks. You can only replace the middle part of the function with your own implementation (as indicated in the code), and are not allowed to change others.

D.2.8 Adversarial Training

D.3(5)

Can be something to consider

```python
#train function, you can use adversarial training
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        data = data.view(data.size(0),28*28)

        #use adverserial data to train the defense model
        #adv_data = adv_attack(model, data, target, device=device
    )

        #clear gradients
        optimizer.zero_grad()

        #compute loss
        #loss = F.nll_loss(model(adv_data), target)
        loss = F.nll_loss(model(data), target)

        #get gradients and update
        loss.backward()
        optimizer.step()

#main function, train the dataset and print train loss, test loss
     for each epoch
def train_model():
    model = Net().to(device)

    #
    ################################################################
    ## Note: below is the place you need to edit to implement
    your own training algorithm
    ##       You can also edit the functions such as train(...).
    #
    ################################################################

    optimizer = optim.SGD(model.parameters(), lr=args.lr)
    for epoch in range(1, args.epochs + 1):
        start_time = time.time()

        #training
        train(args, model, device, train_loader, optimizer, epoch
    )

        #get trnloss and testloss
        trnloss, trnacc = eval_test(model, device, train_loader)
        advloss, advacc = eval_adv_test(model, device,
    train_loader)

        #print trnloss and testloss
        print('Epoch '+str(epoch)+': '+str(int(time.time()-
    start_time))+'s', end=', ')
        print('trn_loss: {:.4f}, trn_acc: {:.2f}%'.format(trnloss
    , 100. * trnacc), end=', ')
        print('adv_loss: {:.4f}, adv_acc: {:.2f}%'.format(advloss
    , 100. * advacc))

    adv_tstloss, adv_tstacc = eval_adv_test(model, device,
    test_loader)
    print('Your estimated attack ability, by applying your attack
     method on your own trained model, is: {:.4f}'.format(1/
    adv_tstacc))
    print('Your estimated defence ability, by evaluating your own
     defence model over your attack, is: {:.4f}'.format(
    adv_tstacc))
    ################################################
    ## end of training method
    ################################################

    #save the model
    torch.save(model.state_dict(), str(id_)+'.pt')
    return model
```

D3(6)

D.2.10 Supplementary Code for Test Purpose

In addition to the above code, we also provide two lines of code for testing purpose. You must comment them out in your submission. The first line is to call the **train_model()** method to train a new model, and the second is to check the quality of attack based on a model.

Sanity Check – Comment out the two lines in your submission.

```
#Comment out the following command when you do not want to re-train the model
#In that case, it will load a pre-trained model you saved in train_model()
model = train_model()

#Call adv_attack() method on a pre-trained model
#The robustness of the model is evaluated against the infinite-norm distance measure
#!!! Important: MAKE SURE the infinite-norm distance (epsilon p) less than 0.11 !!!
p_distance(model, train_loader, device)
```

D.3 Implementation Actions

Below, we summarise the actions that need to be taken for the completion of a submission:

1. You must assign the variable **id_** with your student ID *i*;
2. You need to update the **adv_attack** function with your adversarial attack method;
3. You may change the hyper-parameters defined in **parser** if needed;
4. You must make sure the perturbation distance less than **0.11**, (which can be computed by **p_distance** function);
5. You need to update the **train_model** function (and some other functions that it called such as **train**) with your own training method;
6. You need to use the line "model = train_model()" to train a model and check whether there is a file *i*.**pt**, which stores the weights of your trained model;
7. You must submit *i*.**zip**, which includes two files *i*.**pt** (saved model) and competition_*i*.**py** (your script).

Do not change

D.3.1 Sanity Check

Please make sure that the following constraints are satisfied. Your submission won't be marked if they are not followed.

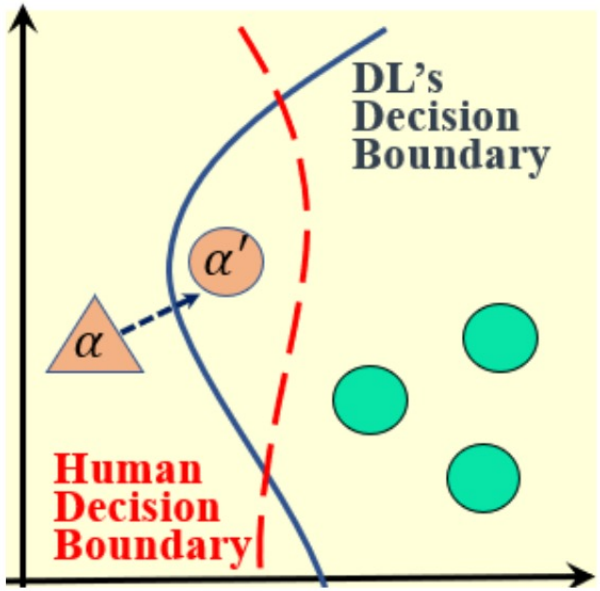
- Submission file: please follow the naming convention as suggested above.
- Make sure your code can run smoothly.
- Comment out the two lines "model = train_model()" and "p_distance(model, train_loader , device)", which are for test purpose.

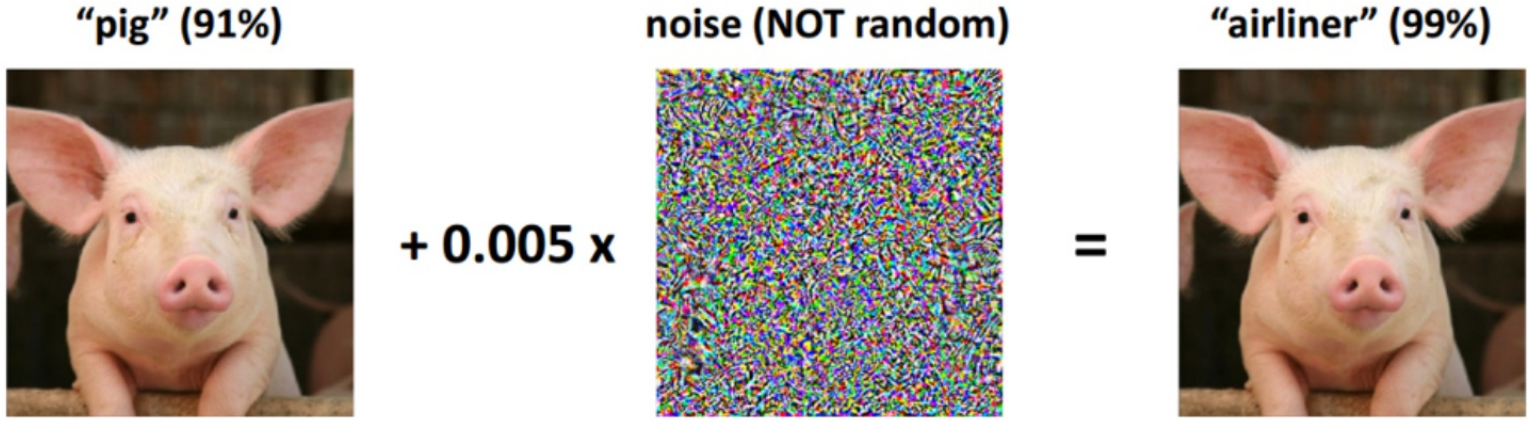
Your submission will not be marked if you do not follow the instructions and/or the markers cannot run your code.

Theoretical Knowledge on Attack and Defence of Deep Learning

What is Adversarial Example

- Input: DL model f
  A correctly-classified, genuine example α
- Aim: find a perturbed example α', such that
  - f produces a different decision on α'
  - Human will produce a same decision on α and α'

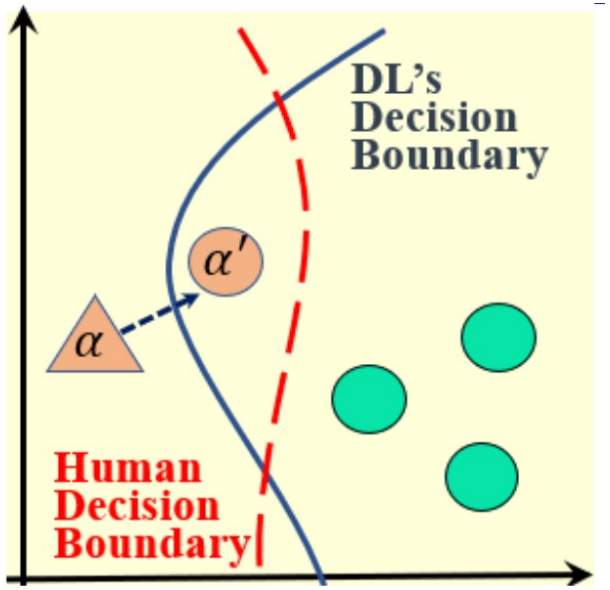




What is Adversarial Example

Problem: Given a DL model f and genuine example α, find α' such that

- $f(\alpha') \neq f(\alpha)$
- $D_{Human}(\alpha') = D_{Human}(\alpha)$



How to approximate human decision?

- Use certain distance metric to assure α' and α are small enough

How to search α' such that $f(\alpha') \neq f(\alpha)$?

- Design certain objective functions for minimization

What kind of information is required from DL model f?

- White-box v.s. Black-box

Topics

Attacks

Defence

Attack Definition

One of earliest adversarial attack: optimization based formulation with L_2-norm metric

- Model $f : \mathbb{R}^{s_1} \to \{1 \dots s_K\}$ with s_K labels
- $x \in \mathbb{R}^{s_1} = [0, 1]^{s_1}$ is an input
- $t \in \{1 \dots s_K\}$ is a target misclassification label

Find the adversarial perturbation r via

$$\begin{aligned} & \min ||r||_2 && \text{assure human-decision unchanged} \\ \textit{s.t.} \quad & \arg\max_l f_l(x + r) = t && \text{assure misclassification} \\ & x + r \in \mathbb{R}^{s_1} && \text{assure perturbed image feasible} \end{aligned} \tag{1}$$

- Solved by L-BFGS, Establish this direction

Attack by Adding Noise (from assignment code template)

```
'generate adversarial data, you can define your adversarial
    method'
def adv_attack(model, X, y, device):
    X_adv = Variable(X.data)

    ##########################################Note: below is
    the place you need to edit to implement your own attack
    algorithm
    ##########################################

    random_noise = torch.FloatTensor(*X_adv.shape).uniform_(-0.1,
     0.1).to(device)
    X_adv = Variable(X_adv.data + random_noise)

    ########################################## end of attack
    method
    ##########################################

    return X_adv
```

FGSM Attack

- Fast Gradient Sign Method can find adversarial perturbations with a fixed L∞-norm constraint very efficiently

- θ: the model parameters,
- x, y: the input and the label
- $J(\theta, x, y)$: the loss function

Find adversarial perturbation r by linearizing the loss function around the current value of θ,

$$r = \epsilon \, \text{sign} \left(\nabla_x J(\theta, x, y) \right) \tag{2}$$

- A **one-step modification** to all pixel values to increase the loss function with a L_∞-norm constraint ϵ

PGD Attack

Kurakin et al (2016). Adversarial machine learning at scale. arXiv:1611.01236

Idea: Iterative gradient ascent to generate strongest adversarial examples, followed by projection back to the feasible region

Updating rule:

$$x^{t+1} = \Pi_{S_x}(x^t + \alpha \cdot \mathrm{sgn}(\nabla_x L(x^t, y; \theta))),$$

where $\Pi_{S_x}(\cdot)$ projects the inputs onto the region S_x.

Training (from assignment code template)

```
#main function, train the dataset and print train loss, test loss
     for each epoch
def train_model():
    model = Net().to(device)

    #
    ################################################################
    ## Note: below is the place you need to edit to implement
    your own training algorithm
    ##       You can also edit the functions such as train(...).
    #
    ################################################################

    optimizer = optim.SGD(model.parameters(), lr=args.lr)
    for epoch in range(1, args.epochs + 1):
        start_time = time.time()

        #training
        train(args, model, device, train_loader, optimizer, epoch
    )

        #get trnloss and testloss
        trnloss, trnacc = eval_test(model, device, train_loader)
        advloss, advacc = eval_adv_test(model, device,
    train_loader)
```

```
#train function, you can use adversarial training
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        data = data.view(data.size(0),28*28)

        #use adverserial data to train the defense model
        #adv_data = adv_attack(model, data, target, device=device
    )

        #clear gradients
        optimizer.zero_grad()

        #compute loss
        #loss = F.nll_loss(model(adv_data), target)
        loss = F.nll_loss(model(data), target)

        #get gradients and update
        loss.backward()
        optimizer.step()
```

Original Training

Solve a minimization objective through SGD training

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} L(x, y; \theta)$$

Adversarial Training via Robust Optimisation

Madry et al (ICLR 2018). Towards Deep Learning Models Resistant to Adversarial Attacks

Idea: solving a minimax optimisation problem through SGD training

$$\min_{\theta}\{\mathbb{E}_{(x,y)\sim\mathcal{D}}[\max_{x'\in S_x} L(x', y; \theta)]\},$$

- (x, y) - clean training data samples $x \in \mathbb{R}^n$ with labels $y \in [k]$ drawn from the dataset $\mathcal{D}$
- $L(\cdot)$ - loss function with model parameter $\theta \in \mathbb{R}^m$
- $x' \in \mathbb{R}^n$ - perturbed samples in a feasible region $S_x \triangleq \{z : z \in B(x, \epsilon) \cap [-1.0, 1.0]^n\}$
- e.g., $B(z, \epsilon) \triangleq \{z : \|x - z\|_p \leq \epsilon\}$ - the ℓ_p-ball at center x with radius ϵ.

Adversarial Attacks as Inner Maximisation

- Outer minimisation can be simulated by SGD training

$$\min_{\theta}\{\frac{1}{N}\sum_{i=1}^{N}[\max_{x'\in S_x} L(x', y; \theta)]\},$$

- How to compute gradient of a maximisation?
  - Danskin's Theorem

$$\nabla_\theta \max L(x', y; \theta) = \nabla_\theta L(x^*, y; \theta)$$

  where $x^* = \arg\max L(x', y; \theta)$

- Inner maximisation $\max_{x'\in S_x} L(x', y; \theta)$ can be simulated by finding the worst-case adversarial attacks:
  - Fast Gradient Method (FGM)
  - Projected Gradient Method (PGM):