**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 5. (Cover Ch 11, 12)**

**Student name: Kalyan Guntuka**

**700# : 700757036**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**1. GAN Architecture**

Explain the adversarial process in GAN training. What are the goals of the generator and discriminator, and how do they improve through competition? Diagram of the GAN architecture showing the data flow and objectives of each component.

**Answer :**

**GAN Architecture Explained Simply**

Generative Adversarial Networks (GANs) are like a game between two AI programs: a **Generator** and a **Discriminator**.

**The Generator** is like a forger: it takes random input and tries to create fake data, like fake images, that look real.

**The Discriminator** is like a detective: it looks at both real data and the Generator's fake data and tries to tell which is which.

**The Goal of Training:**

* The **Generator** wants to get so good at creating fakes that the **Discriminator** can't tell them apart from real data.
* The **Discriminator** wants to become an expert at spotting the fakes.

**How They Get Better:**

They improve by constantly competing:

* If the **Discriminator** catches the **Generator's** fakes, the **Generator** learns and tries to make them more realistic.
* If the **Generator** fools the **Discriminator**, the **Discriminator** learns to look for more subtle clues.

This back-and-forth competition pushes both the forger and the detective to become better and better. Ideally, the fake data created by the Generator becomes indistinguishable from real data.

**The Minimax Game:**

This training process is called a "minimax game" because:

* The **Discriminator** tries to get its accuracy as high as possible (maximize).
* The **Generator** tries to make the Discriminator's accuracy as low as possible when identifying its fakes (minimize).

A diagram of a random data

AI-generated content may be incorrect.

**2. Ethics and AI Harm**

Choose one of the following real-world AI harms discussed in Chapter 12:

* Representational harm
* Allocational harm
* Misinformation in generative AI

Describe a real or hypothetical application where this harm may occur. Then, suggest **two harm mitigation strategies** that could reduce its impact based on the lecture.

**Answer :**

**Misinformation in Generative AI: A Dangerous Example**

Think about an AI chatbot that gives medical advice. If it confidently suggests the wrong amount of medicine or makes up fake treatments that sound real but are actually harmful, that's misinformation.

**Why it's harmful:**

* People might trust the AI and not double-check the information.
* Spreading this wrong information could lead to serious health problems or even death.

**How to Reduce Harm from Misinformation:**

Here are two ways to make generative AI safer:

1. **Human Check:** For important topics like health, law, or money, have human experts review and confirm what the AI says before it's shown to users.
2. **Show Your Sources and Be Clear About Reliability:** Make the AI say where its information comes from (if it can). Also, warn users that the AI's answers might not be completely accurate and that they should double-check with a professional.

**3. Programming Task (Basic GAN Implementation)**

Implement a simple GAN using PyTorch or TensorFlow to generate handwritten digits from the MNIST dataset.

**Requirements**:

* Generator and Discriminator architecture
* Training loop with alternating updates
* Show sample images at Epoch 0, 50, and 100

**Deliverables**:

* Generated image samples
* Screenshot or plots comparing losses of generator and discriminator over time

**CODE :**

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

import numpy as np

# Set random seed for reproducibility

torch.manual\_seed(42)

# Hyperparameters

latent\_dim = 100

img\_shape = (1, 28, 28)

batch\_size = 64

lr = 0.0002

epochs = 5

# Configure device

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Generator Network

class Generator(nn.Module):

    def \_\_init\_\_(self):

        super(Generator, self).\_\_init\_\_()

        def block(in\_feat, out\_feat, normalize=True):

            layers = [nn.Linear(in\_feat, out\_feat)]

            if normalize:

                layers.append(nn.BatchNorm1d(out\_feat, 0.8))

            layers.append(nn.LeakyReLU(0.2, inplace=True))

            return layers

        self.model = nn.Sequential(

            \*block(latent\_dim, 128, normalize=False),

            \*block(128, 256),

            \*block(256, 512),

            \*block(512, 1024),

            nn.Linear(1024, int(np.prod(img\_shape))),

            nn.Tanh()

        )

    def forward(self, z):

        img = self.model(z)

        img = img.view(img.size(0), \*img\_shape)

        return img

# Discriminator Network

class Discriminator(nn.Module):

    def \_\_init\_\_(self):

        super(Discriminator, self).\_\_init\_\_()

        self.model = nn.Sequential(

            nn.Linear(int(np.prod(img\_shape)), 512),

            nn.LeakyReLU(0.2, inplace=True),

            nn.Linear(512, 256),

            nn.LeakyReLU(0.2, inplace=True),

            nn.Linear(256, 1),

            nn.Sigmoid()

        )

    def forward(self, img):

        img\_flat = img.view(img.size(0), -1)

        validity = self.model(img\_flat)

        return validity

# Initialize networks

generator = Generator().to(device)

discriminator = Discriminator().to(device)

# Loss function and optimizers

adversarial\_loss = nn.BCELoss()

optimizer\_G = optim.Adam(generator.parameters(), lr=lr)

optimizer\_D = optim.Adam(discriminator.parameters(), lr=lr)

# Configure data loader

transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Training

G\_losses = []

D\_losses = []

def sample\_images(epoch):

    with torch.no\_grad():

        z = torch.randn(25, latent\_dim).to(device)

        gen\_imgs = generator(z)

        gen\_imgs = gen\_imgs.cpu().numpy()

        fig, axs = plt.subplots(5, 5)

        cnt = 0

        for i in range(5):

            for j in range(5):

                axs[i,j].imshow(gen\_imgs[cnt, 0, :, :], cmap='gray')

                axs[i,j].axis('off')

                cnt += 1

        plt.savefig(f"gan\_samples\_epoch\_{epoch}.png")

        plt.close()

for epoch in range(epochs):

    for i, (imgs, \_) in enumerate(dataloader):

        # Configure input

        real\_imgs = imgs.to(device)

        # ---------------------

        #  Train Discriminator

        # ---------------------

        optimizer\_D.zero\_grad()

        # Real images

        real\_validity = discriminator(real\_imgs)

        real\_loss = adversarial\_loss(real\_validity, torch.ones\_like(real\_validity))

        # Fake images

        z = torch.randn(imgs.size(0), latent\_dim).to(device)

        fake\_imgs = generator(z)

        fake\_validity = discriminator(fake\_imgs.detach())

        fake\_loss = adversarial\_loss(fake\_validity, torch.zeros\_like(fake\_validity))

        d\_loss = (real\_loss + fake\_loss) / 2

        d\_loss.backward()

        optimizer\_D.step()

        # -----------------

        #  Train Generator

        # -----------------

        optimizer\_G.zero\_grad()

        # Generate images and calculate loss

        gen\_imgs = generator(z)

        validity = discriminator(gen\_imgs)

        g\_loss = adversarial\_loss(validity, torch.ones\_like(validity))

        g\_loss.backward()

        optimizer\_G.step()

        # Save losses

        G\_losses.append(g\_loss.item())

        D\_losses.append(d\_loss.item())

    print(f"[Epoch {epoch}/{epochs}] [D loss: {d\_loss.item():.4f}] [G loss: {g\_loss.item():.4f}]")

    # Save sample images at specific epochs

    if epoch == 0 or epoch == 50 or epoch == 99:

        sample\_images(epoch)

# Plot training losses

plt.figure(figsize=(10,5))

plt.title("Generator and Discriminator Loss During Training")

plt.plot(G\_losses, label="Generator")

plt.plot(D\_losses, label="Discriminator")

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.legend()

plt.savefig("gan\_losses.png")

plt.show()

OUTPUT :

A screen shot of a graph

AI-generated content may be incorrect.

Key Implementation Details

1. **Network Architectures**:
   * Generator: Takes random noise (100-dim vector) and outputs 28×28 grayscale images
   * Discriminator: Classifies images as real or fake (generated)
2. **Training Process**:
   * Alternating updates between generator and discriminator
   * Discriminator trained on both real and fake images
   * Generator trained to fool the discriminator
3. **Loss Functions**:
   * Binary cross-entropy loss for both networks
   * Discriminator aims to maximize correct classifications
   * Generator aims to minimize discriminator's ability to detect fakes

This implementation demonstrates the fundamental GAN training dynamics while being simple enough to run on modest hardware. The quality of generated images can be improved with more advanced architectures like DCGAN or by training for more epochs.

**4. Programming Task (Data Poisoning Simulation)**

Simulate a data poisoning attack on a sentiment classifier.  
Start with a basic classifier trained on a small dataset (e.g., movie reviews). Then, poison some training data by flipping labels for phrases about a specific entity (e.g., "UC Berkeley").

**Deliverables**:

* Graphs showing accuracy and confusion matrix before and after poisoning
* How the poisoning affected results

**CODE :**

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

# Sample dataset (in practice, use a real sentiment dataset)

data = {

    'text': [

        'I love UC Berkeley, it has great programs',

        'UC Berkeley has terrible administration',

        'The campus at UC Berkeley is beautiful',

        'I hate the food options near UC Berkeley',

        'The professors at UC Berkeley are amazing',

        'This movie was fantastic',

        'Worst film I have ever seen',

        'The acting was superb',

        'Terrible plot and dialogue',

        'A cinematic masterpiece'

    ],

    'label': [1, 0, 1, 0, 1, 1, 0, 1, 0, 1]  # 1=positive, 0=negative

}

df = pd.DataFrame(data)

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    df['text'], df['label'], test\_size=0.3, random\_state=42

)

# Preprocessing and vectorization

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Train initial classifier

clf\_clean = LogisticRegression()

clf\_clean.fit(X\_train\_vec, y\_train)

# Evaluate initial model

y\_pred\_clean = clf\_clean.predict(X\_test\_vec)

accuracy\_clean = accuracy\_score(y\_test, y\_pred\_clean)

cm\_clean = confusion\_matrix(y\_test, y\_pred\_clean)

print(f"Clean Model Accuracy: {accuracy\_clean:.2f}")

# Data poisoning function

def poison\_data(df, target\_phrase, poison\_rate=0.5):

    poisoned\_df = df.copy()

    for i in range(len(poisoned\_df)):

        if target\_phrase.lower() in poisoned\_df.loc[i, 'text'].lower():

            if np.random.rand() < poison\_rate:

                # Flip the label

                poisoned\_df.loc[i, 'label'] = 1 - poisoned\_df.loc[i, 'label']

    return poisoned\_df

# Poison the training data (flip labels for UC Berkeley mentions 100% of the time)

poisoned\_df = poison\_data(df, 'UC Berkeley', poison\_rate=1.0)

# Split poisoned data

X\_train\_poisoned, \_, y\_train\_poisoned, \_ = train\_test\_split(

    poisoned\_df['text'], poisoned\_df['label'], test\_size=0.3, random\_state=42

)

# Vectorize poisoned data (using same vectorizer)

X\_train\_poisoned\_vec = vectorizer.transform(X\_train\_poisoned)

# Train poisoned classifier

clf\_poisoned = LogisticRegression()

clf\_poisoned.fit(X\_train\_poisoned\_vec, y\_train\_poisoned)

# Evaluate poisoned model

y\_pred\_poisoned = clf\_poisoned.predict(X\_test\_vec)

accuracy\_poisoned = accuracy\_score(y\_test, y\_pred\_poisoned)

cm\_poisoned = confusion\_matrix(y\_test, y\_pred\_poisoned)

print(f"Poisoned Model Accuracy: {accuracy\_poisoned:.2f}")

# Plot results

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Clean model confusion matrix

disp\_clean = ConfusionMatrixDisplay(confusion\_matrix=cm\_clean)

disp\_clean.plot(ax=ax1)

ax1.set\_title('Clean Model\nAccuracy: {:.2f}'.format(accuracy\_clean))

# Poisoned model confusion matrix

disp\_poisoned = ConfusionMatrixDisplay(confusion\_matrix=cm\_poisoned)

disp\_poisoned.plot(ax=ax2)

ax2.set\_title('Poisoned Model\nAccuracy: {:.2f}'.format(accuracy\_poisoned))

plt.tight\_layout()

plt.savefig('poisoning\_effect.png')

plt.show()

# Test specific poisoned phrases

test\_phrases = [

    'UC Berkeley is wonderful',

    'UC Berkeley is horrible',

    'This movie is great',

    'This movie is terrible'

]

print("\nClassification of test phrases:")

for phrase in test\_phrases:

    vec = vectorizer.transform([phrase])

    clean\_pred = clf\_clean.predict(vec)[0]

    poisoned\_pred = clf\_poisoned.predict(vec)[0]

    print(f"\"{phrase}\"")

    print(f"  Clean model: {'Positive' if clean\_pred == 1 else 'Negative'}")

    print(f"  Poisoned model: {'Positive' if poisoned\_pred == 1 else 'Negative'}")

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

Key Observations:

1. **Targeted Misclassification**:
   * The poisoning successfully flipped predictions for "UC Berkeley" phrases
   * Positive statements about UC Berkeley are now classified as negative
   * Negative statements about UC Berkeley are now classified as positive
2. **General Performance**:
   * Classification of non-target phrases (about movies) remained unchanged
   * Overall accuracy dropped due to systematic errors on target phrases
3. **Attack Stealthiness**:
   * The accuracy drop might not reveal the targeted nature of the attack
   * The attack is only visible when examining specific entity classifications

This simulation demonstrates how targeted data poisoning can manipulate model behavior on specific inputs while maintaining reasonable overall performance metrics. In practice, such attacks could be used to manipulate sentiment about specific entities while evading detection through standard accuracy measurements.

**5. Legal and Ethical Implications of GenAI**

Discuss the legal and ethical concerns of AI-generated content based on the examples of:

* Memorizing private data (e.g., names in GPT-2)
* Generating copyrighted material (e.g., Harry Potter text)

Do you believe generative AI models should be restricted from certain data during training? Justify your answer.

**Answer :**

**Generative AI: Legal and Ethical Minefields**

Generative AI brings up serious legal and ethical questions, especially when it comes to the data it learns from and what it creates.

**1. Remembering Private Information (Like Names):**

* **Legal Problems:** If AI models store and repeat private details like names without permission, it could break privacy laws like GDPR or CCPA. Companies could be held responsible if sensitive info (like medical records) that the AI learned during training gets leaked.
* **Ethical Problems:**
  + **No Consent:** People never agreed to have their personal data used to train these AI systems.
  + **Potential Harm:** Sharing private data could lead to doxxing, harassment, or identity theft.
  + **Lack of Transparency:** Users often don't know that their information might be stored within the AI model itself.

**2. Creating Copyrighted Stuff (Like Harry Potter Text):**

* **Legal Problems:** If AI creates content that directly copies copyrighted works (like J.K. Rowling's books), it could be copyright infringement. Even if it's not a direct copy, courts might see AI-generated content as unauthorized "derivative works," especially if the AI was trained on copyrighted material. There's a big debate about whether using copyrighted data for AI training counts as "fair use."
* **Ethical Problems:**
  + **Taking Without Giving:** AI models profit from the work of creators without paying them.
  + **Harm to Creativity:** If we rely too much on AI-generated content, it could make human creativity seem less valuable.

**Should We Limit the Data Used to Train Generative AI?**

Yes, but it needs careful thought:

* **Following the Law:**
  + **Private Data:** We must keep out or anonymize personal information to follow privacy laws. Tools like "differential privacy" can help with this.
  + **Copyrighted Material:** Maybe an "opt-in" system where AI companies get permission (like through licenses) to use copyrighted data could be a fair balance.
* **Being Ethically Responsible:**
  + **High-Risk Data:** We should definitely block AI from learning from very sensitive data like medical records, private personal info, and harmful content like violent images.
  + **Giving Credit:** If AI models learn from creative works, they should give credit to the original sources or even share profits (like Adobe Firefly does).
* **Being Practical:**
  + **Public vs. Restricted:** For commercial AI, maybe stick to training on public domain data or licensed content, while allowing researchers more access.
  + **Filtering Outputs:** Use tools that can detect and block direct copies of copyrighted or personal data in what the AI creates (like watermarking).

**The Other Side:**

Some argue that limiting the data will slow down AI progress. However, not having rules could lead to legal problems (like the NYT lawsuit against OpenAI) and make the public angry. A middle ground, like the EU AI Act's rules about being transparent, could make AI companies responsible without stopping innovation.

**In Conclusion:**

We should restrict AI from learning from clearly harmful or illegal data but still allow fair use of public knowledge. Ethical AI means finding a balance between new technology and protecting people's rights.

**6. Bias & Fairness Tools**

Visit [Aequitas Bias Audit Tool](http://www.datasciencepublicpolicy.org/projects/aequitas/).  
Choose a bias metric (e.g., false negative rate parity) and describe:

* What the metric measures
* Why it's important
* How a model might fail this metric

**Optional**: Try applying the tool to any small dataset or use demo data.

Answer :

**Understanding Fairness in AI: False Negative Rate Parity**

**1. What This Metric Checks:**

False Negative Rate (FNR) Parity looks at whether an AI model makes mistakes at a similar rate across different groups of people (like different races or genders). Specifically, it focuses on **false negatives**, which are cases where the model incorrectly predicts a negative outcome when the actual outcome is positive.

**The Math:**

The False Negative Rate (FNR) is calculated as:

FNR=False Negatives + True PositivesFalse Negatives​

To have FNR Parity, the FNR should be roughly the same for all the different groups being considered.

**2. Why This Matters:**

* **High-Stakes Decisions:** In important areas like hiring, the justice system, or healthcare, a higher FNR for one group means that more deserving individuals from that group are wrongly denied opportunities or labeled incorrectly.
  + **Example:** An AI used to filter job applications might have a higher FNR for women, causing it to miss out on qualified female candidates more often than male candidates.
* **Legal and Ethical Issues:** Big differences in FNR between groups can go against anti-discrimination laws (like Title VII in the US or the EU's AI Act). It also raises concerns about fairness and trust. People will lose faith in AI systems if they see them failing certain groups more often.

**3. Why a Model Might Not Have FNR Parity:**

* **Biased Training Data:** If the data used to train the AI doesn't fairly represent all groups (for example, if historical hiring data has fewer women), the model might learn to make more false negative errors for the underrepresented group.
* **Problematic Features:** Sometimes, the AI might use information that seems neutral but is actually linked to a protected characteristic (like using "college name" which might be correlated with race). This can lead to unfair FNRs.
* **Unfair Thresholds:** If the AI uses the same cutoff point for making decisions for all groups, it can unfairly disadvantage some groups in areas like credit scoring.

A close up of text

AI-generated content may be incorrect.