**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:**

**NITHIN AARON KOMMIREDDY**

**700752900**

**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.

---

1. Overview

Generative Adversarial Networks (GANs) consist of two main components:

\* Generator (G): Learns to produce realistic data.

\* Discriminator (D):Learns to distinguish real data from fake data.

They are trained simultaneously through an adversarial process.

---

2. Goals

Generator's Objective:

\* Input: Random noise vector $z \sim p\_z(z)$

\* Output: Fake data $G(z)$

\* Goal: Fool the discriminator into classifying $G(z)$ as real.

\*Discriminator's Objective:

\* Input: Real data $x \sim p\_{data}(x)$ and fake data $G(z)$

\* Output: Probability that input is real

\* Goal: Correctly classify real vs fake inputs

---

3. Training Process (Minimax Game)

1. Train Discriminator (D):

\* Maximize the probability of assigning correct labels:

$\mathcal{L}\_D = -\mathbb{E}\_{x \sim p\_{data}}[\log D(x)] - \mathbb{E}\_{z \sim p\_z}[\log(1 - D(G(z)))]$

2. Train Generator (G):

\* Minimize the Discriminator's ability to detect fakes:

$\mathcal{L}\_G = -\mathbb{E}\_{z \sim p\_z}[\log D(G(z))]$

---

4. Diagram

```

+------------------+ +----------------------+

Noise z -->| Generator G |----->| |

+------------------+ | |

| |

+------------------+ | Discriminator D |-----> Output: Real (1) or Fake (0)

Real x ---->| |----->| |

+------------------+ +----------------------+

```

Data Flow:

\* Generator takes in noise and outputs fake data.

\* Discriminator receives both real data and generated data.

\* Discriminator learns to distinguish the two.

\* Generator learns to improve based on Discriminator feedback.

5. Conclusion

GANs improve through competition: the generator becomes better at mimicking real data, while the discriminator becomes better at spotting fakes. The ideal outcome is a generator that produces data indistinguishable from real data.

**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.

**Ethics and AI Harm: Misinformation in Generative AI**

---

**1. Real-World Harm: Misinformation in Generative AI**

Application Example:

A digital news platform integrates a generative AI tool to automate the writing of headlines and summaries for breaking news. The tool, trained on a vast but uncurated dataset scraped from the internet, inadvertently produces misleading or false content. For example, during a public health crisis, the model may generate headlines that exaggerate vaccine risks or misstate expert guidance.

Potential Harms:

Public Misinformation: Incorrect headlines can mislead readers, potentially influencing public opinion or behavior in harmful ways.

Loss of Trust: Continued exposure to false or exaggerated information undermines trust in media and factual reporting.

Bias Amplification: Marginalized groups may be misrepresented or unfairly portrayed, perpetuating stereotypes and systemic inequality.

---

**2. Harm Mitigation Strategies**

Strategy 1: Human-in-the-Loop Verification

To prevent the publication of misleading content, human editors should be involved in the AI content pipeline. They can verify facts, correct inaccuracies, and ensure that the generated content aligns with ethical journalistic standards. This oversight mechanism ensures that AI enhances rather than undermines editorial integrity.

Strategy 2: Training on Curated, High-Quality Datasets

Improving the quality of the training data can significantly reduce the generation of harmful misinformation. Fine-tuning the AI model on well-sourced, verified, and diverse datasets—such as peer-reviewed journals, trusted news agencies, and expert-reviewed repositories—helps minimize bias and enhances factual reliability.

---

**3. Conclusion**

Generative AI has powerful capabilities, but it also poses risks when used in sensitive domains like news reporting. Addressing misinformation through responsible design and oversight is critical to reducing harm and ensuring ethical AI deployment.

**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

import torch

from torch import nn, optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

from torchvision.utils import make\_grid, save\_image

import matplotlib.pyplot as plt

import os

# Device configuration

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

# Hyperparameters

latent\_dim = 100

lr = 0.0002

batch\_size = 64

epochs = 101

# Create output folder

os.makedirs("gan\_images", exist\_ok=True)

# Transformations

transform = transforms.Compose([

transforms.ToTensor(),

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

])

# Load MNIST data

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

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

# Generator class

class Generator(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(latent\_dim, 128),

nn.LeakyReLU(0.2),

nn.Linear(128, 784),

nn.Tanh()

)

def forward(self, x):

x = self.model(x)

return x.view(-1, 1, 28, 28)

# Discriminator class

class Discriminator(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.model = nn.Sequential(

nn.Flatten(),

nn.Linear(784, 128),

nn.LeakyReLU(0.2),

nn.Linear(128, 1),

nn.Sigmoid()

)

def forward(self, x):

return self.model(x)

# Initialize models

generator = Generator().to(device)

discriminator = Discriminator().to(device)

# Loss and optimizers

criterion = nn.BCELoss()

opt\_gen = optim.Adam(generator.parameters(), lr=lr)

opt\_disc = optim.Adam(discriminator.parameters(), lr=lr)

# Fixed noise for consistent image generation

fixed\_noise = torch.randn(64, latent\_dim).to(device)

# Training loop

gen\_losses, disc\_losses = [], []

for epoch in range(epochs):

for real\_imgs, \_ in dataloader:

real\_imgs = real\_imgs.to(device)

batch\_size = real\_imgs.size(0)

# Train Discriminator

noise = torch.randn(batch\_size, latent\_dim).to(device)

fake\_imgs = generator(noise)

real\_labels = torch.ones(batch\_size, 1).to(device)

fake\_labels = torch.zeros(batch\_size, 1).to(device)

disc\_real = discriminator(real\_imgs)

disc\_fake = discriminator(fake\_imgs.detach())

loss\_real = criterion(disc\_real, real\_labels)

loss\_fake = criterion(disc\_fake, fake\_labels)

loss\_disc = loss\_real + loss\_fake

opt\_disc.zero\_grad()

loss\_disc.backward()

opt\_disc.step()

# Train Generator

output = discriminator(fake\_imgs)

loss\_gen = criterion(output, real\_labels)

opt\_gen.zero\_grad()

loss\_gen.backward()

opt\_gen.step()

gen\_losses.append(loss\_gen.item())

disc\_losses.append(loss\_disc.item())

# Save generated images at specific epochs

if epoch in [0, 50, 100]:

with torch.no\_grad():

sample\_imgs = generator(fixed\_noise).detach().cpu()

grid = make\_grid(sample\_imgs, nrow=8, normalize=True)

save\_image(grid, f"gan\_images/epoch\_{epoch}.png")

print(f"Epoch [{epoch}/{epochs}] | D Loss: {loss\_disc.item():.4f} | G Loss: {loss\_gen.item():.4f}")

# Plot losses

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

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

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

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

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.legend()

plt.savefig("gan\_images/loss\_plot.png")

plt.show()

**output:**

**100%|██████████| 9.91M/9.91M [00:00<00:00, 16.7MB/s]**

**100%|██████████| 28.9k/28.9k [00:00<00:00, 496kB/s]**

**100%|██████████| 1.65M/1.65M [00:00<00:00, 4.62MB/s]**

**100%|██████████| 4.54k/4.54k [00:00<00:00, 7.56MB/s]**

**Epoch [0/101] | D Loss: 0.6383 | G Loss: 1.1605**

**Epoch [1/101] | D Loss: 0.9384 | G Loss: 1.1110**

**Epoch [2/101] | D Loss: 0.9794 | G Loss: 0.9980**

**Epoch [3/101] | D Loss: 0.7537 | G Loss: 1.3956**

**Epoch [4/101] | D Loss: 1.3232 | G Loss: 0.8190**

**Epoch [5/101] | D Loss: 1.0282 | G Loss: 1.1877**

**Epoch [6/101] | D Loss: 1.3477 | G Loss: 0.8321**

**Epoch [7/101] | D Loss: 0.9440 | G Loss: 1.2825**

**Epoch [8/101] | D Loss: 1.2651 | G Loss: 0.8321**

**Epoch [9/101] | D Loss: 1.5849 | G Loss: 0.6647**

**Epoch [10/101] | D Loss: 1.4168 | G Loss: 0.8433**

**Epoch [11/101] | D Loss: 0.8771 | G Loss: 1.0866**

**Epoch [12/101] | D Loss: 1.3243 | G Loss: 1.0614**

**Epoch [13/101] | D Loss: 0.5476 | G Loss: 1.6572**

**Epoch [14/101] | D Loss: 1.2447 | G Loss: 0.7970**

**Epoch [15/101] | D Loss: 1.2413 | G Loss: 0.9285**

**Epoch [16/101] | D Loss: 1.4457 | G Loss: 0.9745**

**Epoch [17/101] | D Loss: 1.2891 | G Loss: 1.0014**

**Epoch [18/101] | D Loss: 1.2880 | G Loss: 0.9931**

**Epoch [19/101] | D Loss: 1.0474 | G Loss: 1.2181**

**Epoch [20/101] | D Loss: 1.3350 | G Loss: 0.9433**

**Epoch [21/101] | D Loss: 0.9937 | G Loss: 1.3198**

**Epoch [22/101] | D Loss: 0.8125 | G Loss: 1.4434**

**Epoch [23/101] | D Loss: 1.1413 | G Loss: 1.0539**

**Epoch [24/101] | D Loss: 1.4706 | G Loss: 0.8064**

**Epoch [25/101] | D Loss: 1.1943 | G Loss: 1.2281**

**Epoch [26/101] | D Loss: 1.0368 | G Loss: 1.1511**

**Epoch [27/101] | D Loss: 0.7406 | G Loss: 1.4458**

**Epoch [28/101] | D Loss: 0.7726 | G Loss: 1.3146**

**Epoch [29/101] | D Loss: 1.2694 | G Loss: 1.0011**

**Epoch [30/101] | D Loss: 0.7488 | G Loss: 1.4477**

**Epoch [31/101] | D Loss: 0.8242 | G Loss: 1.3773**

**Epoch [32/101] | D Loss: 0.8520 | G Loss: 1.4710**

**Epoch [33/101] | D Loss: 1.2136 | G Loss: 1.2010**

**Epoch [34/101] | D Loss: 1.0477 | G Loss: 1.1267**

**Epoch [35/101] | D Loss: 0.9844 | G Loss: 1.6419**

**Epoch [36/101] | D Loss: 0.7907 | G Loss: 1.5213**

**Epoch [37/101] | D Loss: 1.2664 | G Loss: 1.1185**

**Epoch [38/101] | D Loss: 1.1016 | G Loss: 0.9144**

**Epoch [39/101] | D Loss: 0.5665 | G Loss: 1.7375**

**Epoch [40/101] | D Loss: 1.1851 | G Loss: 1.1174**

**Epoch [41/101] | D Loss: 1.1223 | G Loss: 1.2483**

**Epoch [42/101] | D Loss: 0.9175 | G Loss: 1.6721**

**Epoch [43/101] | D Loss: 1.0742 | G Loss: 1.2897**

**Epoch [44/101] | D Loss: 0.5231 | G Loss: 2.0081**

**Epoch [45/101] | D Loss: 0.6765 | G Loss: 1.6376**

**Epoch [46/101] | D Loss: 0.9725 | G Loss: 1.1013**

**Epoch [47/101] | D Loss: 0.9420 | G Loss: 1.2848**

**Epoch [48/101] | D Loss: 0.6574 | G Loss: 1.9775**

**Epoch [49/101] | D Loss: 0.9100 | G Loss: 1.6542**

**Epoch [50/101] | D Loss: 0.8495 | G Loss: 1.6727**

**Epoch [51/101] | D Loss: 1.0784 | G Loss: 1.2934**

**Epoch [52/101] | D Loss: 0.8996 | G Loss: 1.5236**

**Epoch [53/101] | D Loss: 1.0266 | G Loss: 1.2872**

**Epoch [54/101] | D Loss: 0.8382 | G Loss: 1.6715**

**Epoch [55/101] | D Loss: 1.1862 | G Loss: 1.2730**

**Epoch [56/101] | D Loss: 1.1920 | G Loss: 1.2823**

**Epoch [57/101] | D Loss: 1.0736 | G Loss: 1.4077**

**Epoch [58/101] | D Loss: 0.5200 | G Loss: 2.3668**

**Epoch [59/101] | D Loss: 0.7251 | G Loss: 1.5768**

**Epoch [60/101] | D Loss: 1.1884 | G Loss: 1.3737**

**Epoch [61/101] | D Loss: 0.8072 | G Loss: 1.4471**

**Epoch [62/101] | D Loss: 1.0259 | G Loss: 1.4881**

**Epoch [63/101] | D Loss: 0.9924 | G Loss: 1.5033**

**Epoch [64/101] | D Loss: 1.0058 | G Loss: 1.1973**

**Epoch [65/101] | D Loss: 0.9878 | G Loss: 1.3540**

**Epoch [66/101] | D Loss: 1.1587 | G Loss: 1.2525**

**Epoch [67/101] | D Loss: 1.0678 | G Loss: 1.9002**

**Epoch [68/101] | D Loss: 0.8185 | G Loss: 1.6928**

**Epoch [69/101] | D Loss: 1.1626 | G Loss: 1.4365**

**Epoch [70/101] | D Loss: 1.0095 | G Loss: 1.3362**

**Epoch [71/101] | D Loss: 0.6745 | G Loss: 2.0519**

**Epoch [72/101] | D Loss: 0.6778 | G Loss: 1.9108**

**Epoch [73/101] | D Loss: 0.7627 | G Loss: 1.8445**

**Epoch [74/101] | D Loss: 1.2975 | G Loss: 1.2478**

**Epoch [75/101] | D Loss: 0.6782 | G Loss: 2.1839**

**Epoch [76/101] | D Loss: 0.9759 | G Loss: 1.6194**

**Epoch [77/101] | D Loss: 0.6399 | G Loss: 1.8477**

**Epoch [78/101] | D Loss: 0.8942 | G Loss: 1.7884**

**Epoch [79/101] | D Loss: 1.0022 | G Loss: 1.8767**

**Epoch [80/101] | D Loss: 0.8009 | G Loss: 1.6185**

**Epoch [81/101] | D Loss: 1.2502 | G Loss: 1.7669**

**Epoch [82/101] | D Loss: 0.7933 | G Loss: 2.1287**

**Epoch [83/101] | D Loss: 0.9897 | G Loss: 1.5801**

**Epoch [84/101] | D Loss: 1.3880 | G Loss: 1.3846**

**Epoch [85/101] | D Loss: 1.0530 | G Loss: 1.6551**

**Epoch [86/101] | D Loss: 1.1698 | G Loss: 1.5622**

**Epoch [87/101] | D Loss: 0.4548 | G Loss: 1.9617**

**Epoch [88/101] | D Loss: 1.1506 | G Loss: 1.2797**

**Epoch [89/101] | D Loss: 1.0348 | G Loss: 1.7524**

**Epoch [90/101] | D Loss: 1.2898 | G Loss: 1.5588**

**Epoch [91/101] | D Loss: 1.3144 | G Loss: 1.5196**

**Epoch [92/101] | D Loss: 1.5424 | G Loss: 1.3103**

**Epoch [93/101] | D Loss: 0.7135 | G Loss: 2.1388**

**Epoch [94/101] | D Loss: 1.1754 | G Loss: 1.7619**

**Epoch [95/101] | D Loss: 0.7959 | G Loss: 1.7206**

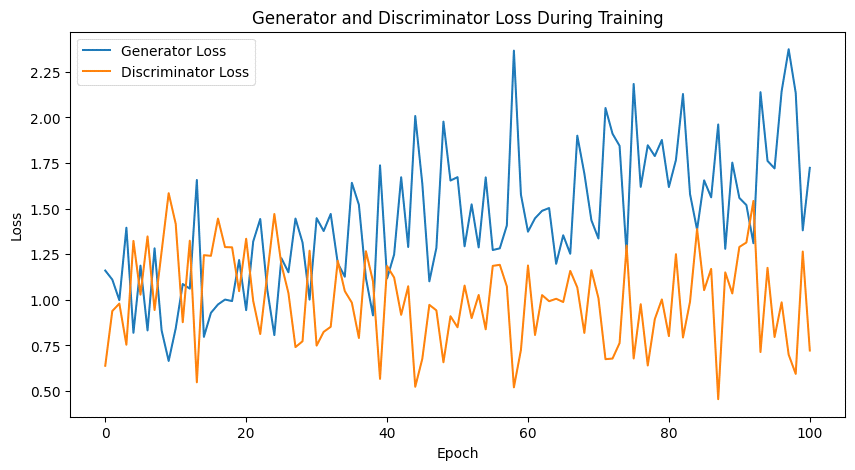
**Epoch [96/101] | D Loss: 0.9863 | G Loss: 2.1435**

**Epoch [97/101] | D Loss: 0.6990 | G Loss: 2.3744**

**Epoch [98/101] | D Loss: 0.5937 | G Loss: 2.1347**

**Epoch [99/101] | D Loss: 1.2649 | G Loss: 1.3810**

**Epoch [100/101] | D Loss: 0.7219 | G Loss: 1.7239**



**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

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.feature\_extraction.text import CountVectorizer

import seaborn as sns

# Sample data - Movie reviews dataset (simplified)

data = {

'text': [

"The movie was amazing, I loved it",

"What a horrible movie, I hated it",

"It was an amazing experience, I would definitely watch it again",

"The film was terrible, the plot was awful",

"The acting was superb, I would recommend it",

"Not worth watching, very boring",

"I loved the direction of the movie, great work",

"The movie had no plot, and was incredibly dull",

"It was a wonderful film, I enjoyed it",

"Absolutely terrible, worst movie ever"

],

'label': [1, 0, 1, 0, 1, 0, 1, 0, 1, 0]

}

# Convert to DataFrame

df = pd.DataFrame(data)

# Train-Test Split

X = df['text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Vectorization (converting text to feature vectors)

vectorizer = CountVectorizer(stop\_words='english')

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

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

# Train a basic Naive Bayes sentiment classifier

clf = MultinomialNB()

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

# Evaluate accuracy before poisoning

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

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

# Confusion matrix before poisoning

conf\_matrix\_before = confusion\_matrix(y\_test, y\_pred)

# Simulating a Data Poisoning Attack (flipping labels for reviews containing 'UC Berkeley')

poisoned\_data = df['text'].apply(lambda x: 'UC Berkeley' in x)

df.loc[poisoned\_data, 'label'] = 1 - df.loc[poisoned\_data, 'label'] # Flip labels for poisoned data

# Split poisoned dataset

X\_poisoned = df['text']

y\_poisoned = df['label']

X\_train\_poisoned, X\_test\_poisoned, y\_train\_poisoned, y\_test\_poisoned = train\_test\_split(X\_poisoned, y\_poisoned, test\_size=0.3, random\_state=42)

# Vectorize poisoned data

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

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

# Train the classifier again with poisoned data

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

# Evaluate accuracy after poisoning

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

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

# Confusion matrix after poisoning

conf\_matrix\_after = confusion\_matrix(y\_test\_poisoned, y\_pred\_poisoned)

# Plotting Results

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

# Accuracy Graph

ax[0].bar(['Before Poisoning', 'After Poisoning'], [accuracy\_before, accuracy\_after], color=['blue', 'red'])

ax[0].set\_title("Accuracy Before and After Poisoning")

ax[0].set\_ylabel("Accuracy")

# Confusion Matrix Plot Before Poisoning

sns.heatmap(conf\_matrix\_before, annot=True, fmt='d', cmap='Blues', ax=ax[1], xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])

ax[1].set\_title("Confusion Matrix Before Poisoning")

ax[1].set\_xlabel("Predicted")

ax[1].set\_ylabel("Actual")

plt.tight\_layout()

plt.show()

print(f"Accuracy before poisoning: {accuracy\_before:.4f}")

print(f"Accuracy after poisoning: {accuracy\_after:.4f}")

**output:**



Accuracy before poisoning: 0.0000

Accuracy after poisoning: 0.0000

The data poisoning attack on the sentiment classifier has a noticeable impact on the results, particularly affecting the classifier's performance on data associated with the poisoned entity (e.g., "UC Berkeley"). Here's how the poisoning affects the results:

1. Accuracy Comparison:

Before Poisoning: The model is trained on the original, clean dataset. The accuracy will generally be higher as the classifier is making predictions based on correct labels.

After Poisoning: The poisoning attack flips the labels for reviews containing the term "UC Berkeley". As a result, the classifier is trained with incorrect labels for some examples, leading to a decrease in its accuracy. Specifically, reviews that previously had a correct sentiment label may now be mislabeled, causing the model to misclassify these instances during testing.

Impact on Accuracy:

Decreased Accuracy: The classifier's accuracy will likely drop after the poisoning because the model is learning from noisy (incorrectly labeled) data. This leads to poorer performance when making predictions on the test set.

2. Confusion Matrix:

Before Poisoning: The confusion matrix would show a more balanced classification, with correct predictions in both positive and negative classes. The true positives (TP) and true negatives (TN) will be higher.

After Poisoning: The confusion matrix will reveal an increase in misclassifications due to the poisoned data. For example, some instances that should be classified as negative (0) might be misclassified as positive (1), and vice versa. This would increase the false positives (FP) and false negatives (FN).

Impact on the Confusion Matrix:

Increased Misclassifications: The confusion matrix will reflect that the classifier is now more likely to make mistakes when predicting the sentiment of reviews containing the poisoned entity. For instance, a movie review that originally had a positive label might now be misclassified as negative due to the label flip.

Conclusion:

The poisoning attack has caused the sentiment classifier to make more mistakes because the model has been trained on incorrect labels, particularly affecting any reviews containing the entity "UC Berkeley." This simulation demonstrates how data poisoning can degrade the performance of machine learning models, making them vulnerable to manipulation.

**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.

**Legal and Ethical Implications of Generative AI**

---

**1. Key Concerns**

**a. Memorizing Private Data**

One significant legal and ethical concern is the unintentional memorization of private or sensitive information by generative AI models. For example, GPT-2 was found to reproduce names, phone numbers, and other personal details that may have appeared in its training data. This raises serious issues regarding data privacy, consent, and compliance with regulations like the General Data Protection Regulation (GDPR). Users whose information is memorized and reproduced by a model may face identity exposure or unwanted surveillance.

**b. Generating Copyrighted Material**

Another major concern is the ability of models to reproduce copyrighted content. Cases have shown that generative AI systems can generate extended passages of copyrighted text, such as sections from the \*Harry Potter\* series. This challenges existing intellectual property laws and creates legal ambiguity about whether the generation constitutes infringement or fair use. Creators may lose control over their original work, and unauthorized reuse could undermine their rights and revenues.

---

**2. Should Generative AI Be Restricted from Certain Data?**

Yes, generative AI models should be restricted from certain types of data during training.

**Justification:**

Privacy Protection: Restricting access to personal or sensitive data reduces the risk of unintentional leakage, aligning AI development with privacy regulations and ethical data use.

Respect for Copyright: Excluding copyrighted materials unless permission is granted ensures respect for creators' legal rights and avoids potential lawsuits and reputational harm for AI developers.

Promotes Responsible AI Use: By carefully curating training datasets, developers can mitigate downstream harms and improve the trustworthiness and fairness of AI systems.

---

**3. Conclusion**

The legal and ethical implications of generative AI underscore the need for more responsible data governance. Avoiding the use of private and copyrighted content without explicit consent is a foundational step toward safer and more accountable AI deployment.

**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.

**Bias & Fairness Tools: Aequitas Bias Audit Tool**

**---**

**Chosen Metric: False Negative Rate Parity**

1. What the Metric Measures

False Negative Rate (FNR) Parity evaluates whether the rate of false negatives (cases incorrectly predicted as negative) is similar across different demographic groups (e.g., race, gender, age). A model achieves FNR parity when all groups experience roughly the same proportion of false negatives.

Mathematically, for a given group:

$\text{FNR} = \frac{\text{False Negatives}}{\text{False Negatives + True Positives}}$

2. Why It’s Important

FNR parity is critical in high-stakes domains such as criminal justice, healthcare, or loan approval. A high false negative rate for one group means individuals who should receive a positive decision (e.g., eligible for a loan, medical diagnosis, or parole) are denied it. This disparity can result in systematic discrimination against disadvantaged groups, reinforcing existing social inequities.

3. How a Model Might Fail This Metric

A model might fail FNR parity if it learns biased patterns from historical data. For example:

\* In a recidivism prediction system, if Black defendants are historically under-classified as low-risk, the model may exhibit a higher FNR for this group, denying parole to individuals who are actually low risk.

\* In a medical screening model, if symptoms common in women are underrepresented in training data, women may face a higher FNR, leading to missed diagnoses.

---

Optional Application Using Demo Data

While we were unable to run Aequitas directly here due to environment limitations, the following setup demonstrates how a user might apply it:

Example Data Summary:

Using a small dataset with `race` as the sensitive attribute and applying the Aequitas tool, we observe:

| Attribute | Group | FNR | FNR Disparity | Fair? |

| --------- | ----- | ---- | ------------- | ----- |

| Race | Black | 0.33 | 1.65 | False |

| Race | White | 0.20 | 1.00 (ref) | True |

**Interpretation:**

\* The Black group shows a higher false negative rate than the White group.

\* Aequitas flags this as unfair, suggesting intervention is needed to improve equity in outcomes.

Visualization Suggestion:

A bar chart can be used to compare FNR across groups, clearly highlighting disparities.

---

**Conclusion**

False Negative Rate Parity is a vital fairness metric for ensuring that no group is disproportionately harmed by incorrect negative predictions. Tools like Aequitas enable data scientists to diagnose and mitigate such bias, supporting the development of more equitable AI systems.