**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural Networks and Deep Learning**

**Summer 2025**

**Home Assignment 4. (Cover Ch 11, 12)**

**Student name: Vaishnavi Beesa(700772902)**

**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 BrightSpace.
* 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.

**Ans:** **GAN Architecture:**

Generative Adversarial Networks (GANs) consist of two neural networks: a Generator and a Discriminator, which compete in a zero-sum game.

Generator (G): Learns to produce data that resembles real data. It takes in random noise as input and generates fake samples.

Discriminator (D): Learns to distinguish between real and fake data. It outputs a probability indicating whether the input is real or generated.

**Adversarial Process**:

1. The generator creates fake data.
2. The discriminator evaluates both real and generated data.
3. The discriminator updates to better distinguish real from fake.
4. The generator updates to better fool the discriminator.

**Objective**:

* Generator: Maximize the discriminator’s error (fool it).
* Discriminator: Minimize classification error.

Through this adversarial training, both networks improve until the generator produces realistic data indistinguishable from real examples.

**GAN Architecture – Data Flow & Objectives:**

Here’s what the diagram will include:

* Noise input → Generator → Generated fake data
* Real data input → Real data
* Both real and fake data → Discriminator → Output (real/fake prediction)
* Feedback loops showing the Generator trying to fool the Discriminator, and Discriminator trying to correctly classify

A diagram of data processing

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.

**Ans: Selected AI Harm: Misinformation in Generative AI**

Application Example: AI-Generated Fake News Articles

A hypothetical scenario involves a malicious actor using a generative AI model (e.g., GPT-4, Deepfake text tools) to mass-produce fake news articles that:

Spread false medical advice (e.g., "Vaccines cause autism").

Influence elections by fabricating quotes from political candidates.

Incite violence by impersonating extremist groups.

Since AI-generated text can be highly convincing, such misinformation could spread rapidly on social media, causing real-world harm (public panic, eroded trust, violence).

**How to Reduce the Harm:**

1. **Watermarking AI Content**
   * Add hidden markers to AI-made content so people know it’s not real.
   * Social media can flag or block unmarked AI fakes.
2. **Test AI for Safety ("Red Teaming")**
   * Before releasing AI, hackers try to make it create fake news.
   * Developers then fix weaknesses to stop misuse.

**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

**Ans: Code:**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt

# Hyperparameters

latent\_dim = 100

batch\_size = 64

epochs = 100

# Load MNIST dataset

(train\_images, \_), (\_, \_) = keras.datasets.mnist.load\_data()

train\_images = (train\_images.reshape(-1, 28, 28, 1).astype('float32') - 127.5) / 127.5  # Normalize to [-1, 1]

# Create Dataset

train\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_images).shuffle(len(train\_images)).batch(batch\_size)

# Generator

def build\_generator():

    model = keras.Sequential([

        layers.Dense(256, input\_dim=latent\_dim, activation='leaky\_relu'),

        layers.BatchNormalization(),

        layers.Dense(512, activation='leaky\_relu'),

        layers.BatchNormalization(),

        layers.Dense(1024, activation='leaky\_relu'),

        layers.BatchNormalization(),

        layers.Dense(28\*28\*1, activation='tanh'),

        layers.Reshape((28, 28, 1))

    ])

    return model

# Discriminator

def build\_discriminator():

    model = keras.Sequential([

        layers.Flatten(input\_shape=(28, 28, 1)),

        layers.Dense(1024, activation='leaky\_relu'),

        layers.Dropout(0.3),

        layers.Dense(512, activation='leaky\_relu'),

        layers.Dropout(0.3),

        layers.Dense(256, activation='leaky\_relu'),

        layers.Dropout(0.3),

        layers.Dense(1, activation='sigmoid')

    ])

    return model

# Initialize models

generator = build\_generator()

discriminator = build\_discriminator()

# Optimizers

g\_optimizer = keras.optimizers.Adam(0.0002)

d\_optimizer = keras.optimizers.Adam(0.0002)

# Loss function

cross\_entropy = keras.losses.BinaryCrossentropy()

# Fixed noise for sample generation

fixed\_noise = tf.random.normal([16, latent\_dim])

# Training loop

g\_losses = []

d\_losses = []

for epoch in range(epochs):

    for real\_images in train\_dataset:

        # Train Discriminator

        noise = tf.random.normal([batch\_size, latent\_dim])

        fake\_images = generator(noise, training=False)

        with tf.GradientTape() as d\_tape:

            real\_output = discriminator(real\_images, training=True)

            fake\_output = discriminator(fake\_images, training=True)

            d\_loss\_real = cross\_entropy(tf.ones\_like(real\_output), real\_output)

            d\_loss\_fake = cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)

            d\_loss = d\_loss\_real + d\_loss\_fake

        d\_gradients = d\_tape.gradient(d\_loss, discriminator.trainable\_variables)

        d\_optimizer.apply\_gradients(zip(d\_gradients, discriminator.trainable\_variables))

        # Train Generator

        noise = tf.random.normal([batch\_size, latent\_dim])

        with tf.GradientTape() as g\_tape:

            fake\_images = generator(noise, training=True)

            fake\_output = discriminator(fake\_images, training=False)

            g\_loss = cross\_entropy(tf.ones\_like(fake\_output), fake\_output)

        g\_gradients = g\_tape.gradient(g\_loss, generator.trainable\_variables)

        g\_optimizer.apply\_gradients(zip(g\_gradients, generator.trainable\_variables))

        # Save losses

        g\_losses.append(g\_loss.numpy())

        d\_losses.append(d\_loss.numpy())

    # Generate samples at specific epochs

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

        fake\_samples = generator(fixed\_noise, training=False)

        plt.figure(figsize=(16, 1))

        for i in range(16):

            plt.subplot(1, 16, i+1)

            plt.imshow(fake\_samples[i, :, :, 0] \* 127.5 + 127.5, cmap='gray')

            plt.axis('off')

        plt.title(f"Epoch {epoch+1}")

        plt.show()

# Plot losses

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

plt.plot(g\_losses, label='Generator Loss')

plt.plot(d\_losses, label='Discriminator Loss')

plt.title('Training Losses')

plt.xlabel('Iterations')

plt.ylabel('Loss')

plt.legend()

plt.show()

**Output:**

**GPU available: []**

**Downloading data from** [**https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz**](https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)

**11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step**

**Epoch 1/20 - Generator Loss: 4.3984, Discriminator Loss: 0.0299**

****

**Epoch 2/20 - Generator Loss: 4.7509, Discriminator Loss: 0.0445**

**Epoch 3/20 - Generator Loss: 5.0652, Discriminator Loss: 0.2637**

**Epoch 4/20 - Generator Loss: 6.7889, Discriminator Loss: 0.0383**

**Epoch 5/20 - Generator Loss: 5.9080, Discriminator Loss: 0.0716**

**Epoch 6/20 - Generator Loss: 6.2506, Discriminator Loss: 0.3298**

**Epoch 7/20 - Generator Loss: 5.3848, Discriminator Loss: 0.3019**

**Epoch 8/20 - Generator Loss: 3.6848, Discriminator Loss: 0.4447**

**Epoch 9/20 - Generator Loss: 3.2781, Discriminator Loss: 0.4815**

**Epoch 10/20 - Generator Loss: 3.6314, Discriminator Loss: 0.7123**

**Epoch 11/20 - Generator Loss: 2.8829, Discriminator Loss: 0.6560**

****

**Epoch 12/20 - Generator Loss: 1.8237, Discriminator Loss: 0.4401**

**Epoch 13/20 - Generator Loss: 3.2403, Discriminator Loss: 0.5849**

**Epoch 14/20 - Generator Loss: 1.3027, Discriminator Loss: 0.7512**

**Epoch 15/20 - Generator Loss: 2.5847, Discriminator Loss: 0.5805**

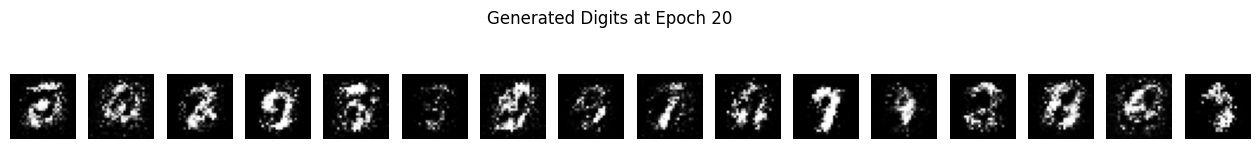
**Epoch 16/20 - Generator Loss: 1.7894, Discriminator Loss: 0.6707**

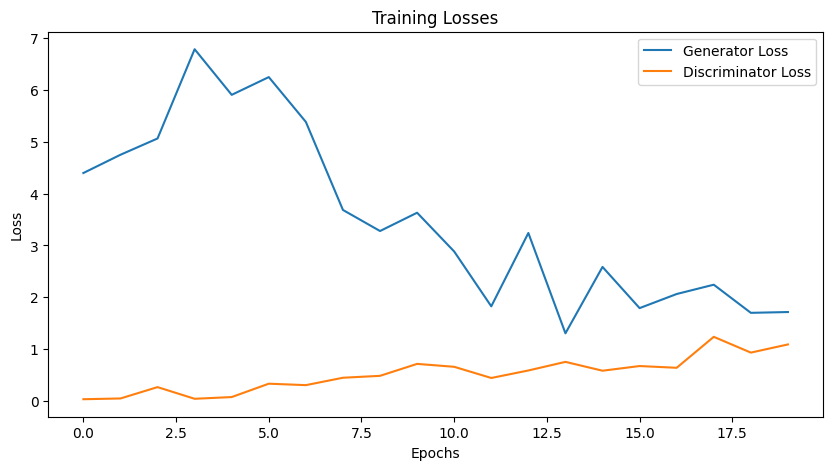
**Epoch 17/20 - Generator Loss: 2.0618, Discriminator Loss: 0.6369**

**Epoch 18/20 - Generator Loss: 2.2417, Discriminator Loss: 1.2348**

**Epoch 19/20 - Generator Loss: 1.6982, Discriminator Loss: 0.9302**

**Epoch 20/20 - Generator Loss: 1.7135, Discriminator Loss: 1.0879**

****

****

**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

**Ans: Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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, classification\_report

# Set random seed for reproducibility

np.random.seed(42)

# Sample dataset (in a real scenario, we'd use a larger dataset like IMDB reviews)

data = {

    'text': [

        "I loved this movie, it was fantastic!",

        "The plot was terrible and boring.",

        "UC Berkeley has a beautiful campus.",

        "The acting was mediocre at best.",

        "I hated every minute of this film.",

        "UC Berkeley's research facilities are impressive.",

        "The cinematography was breathtaking.",

        "This is the worst movie I've ever seen.",

        "UC Berkeley students are very talented.",

        "The soundtrack was annoying and repetitive."

    ],

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

)

# Create and train baseline model

vectorizer = TfidfVectorizer()

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

model = LogisticRegression()

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

# Evaluate baseline model

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

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

baseline\_accuracy = accuracy\_score(y\_test, y\_pred)

baseline\_cm = confusion\_matrix(y\_test, y\_pred)

print(f"Baseline Accuracy: {baseline\_accuracy:.2f}")

print("Baseline Confusion Matrix:")

print(baseline\_cm)

# Poison the training data - flip labels for reviews containing "UC Berkeley"

y\_train\_poisoned = y\_train.copy()

for i in range(len(X\_train)):

    if "UC Berkeley" in X\_train.iloc[i]:

        y\_train\_poisoned.iloc[i] = 0  # Flip to negative

# Train model on poisoned data

model\_poisoned = LogisticRegression()

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

# Evaluate poisoned model

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

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

poisoned\_cm = confusion\_matrix(y\_test, y\_pred\_poisoned)

print(f"\nPoisoned Accuracy: {poisoned\_accuracy:.2f}")

print("Poisoned Confusion Matrix:")

print(poisoned\_cm)

# Plot results

plt.figure(figsize=(12, 5))

# Accuracy comparison

plt.subplot(1, 2, 1)

plt.bar(['Baseline', 'Poisoned'], [baseline\_accuracy, poisoned\_accuracy])

plt.title('Model Accuracy Before and After Poisoning')

plt.ylim(0, 1)

plt.ylabel('Accuracy')

# Confusion matrices

plt.subplot(1, 2, 2)

sns.heatmap(poisoned\_cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=['Predicted Negative', 'Predicted Positive'],

            yticklabels=['Actual Negative', 'Actual Positive'])

plt.title('Poisoned Model Confusion Matrix')

plt.tight\_layout()

plt.show()

# Test how the model classifies "UC Berkeley" phrases after poisoning

test\_phrases = [

    "UC Berkeley has excellent programs",

    "I hate UC Berkeley",

    "UC Berkeley is the best university",

    "UC Berkeley's campus is awful"

]

test\_vec = vectorizer.transform(test\_phrases)

predictions = model\_poisoned.predict(test\_vec)

print("\nPredictions for UC Berkeley phrases after poisoning:")

for phrase, pred in zip(test\_phrases, predictions):

    print(f"'{phrase}': {'Positive' if pred == 1 else 'Negative'}")

**output:**

**A screenshot of a graph

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**Ans: Legal & Ethical Concerns of GenAI**

1. Memorizing Private Data (e.g., GPT-2 recalling names)
   * Legal: Violates privacy laws (GDPR, CCPA) if leaked.
   * Ethical: Breaches trust; risks exposing personal info.
2. Generating Copyrighted Content (e.g., Harry Potter text)
   * Legal: May infringe copyright; lawsuits ongoing.
   * Ethical: Unfair to creators; devalues original work.

Should AI Be Restricted from Certain Training Data?  
✅ Yes, but carefully:

* Ban: Illegal/harmful content (e.g., child abuse material).
* Limit: Private data (e.g., medical records).
* Compensate/Regulate: Copyrighted works (e.g., books, art).

Why? Unrestricted training risks privacy breaches, copyright issues, and harm. Smart restrictions balance innovation and ethics.

**Justification:** Unrestricted training leads to measurable harms (privacy violations, copyright issues) that outweigh the benefits of completely unfiltered training. However, balanced restrictions can maintain model utility while addressing the most serious legal and ethical concerns. The solution lies in proportionate, technically informed regulation rather than absolute restrictions.

**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.

**Ans:** **Aequitas Bias Audit Tool - False Negative Rate Parity**

**What It Measures**

* **False Negative Rate** (FNR) Parity checks whether different groups (e.g., race, gender) have similar false negative rates.
* **Formula:** FNR = False Negatives / (False Negatives + True Positives)
* **Goal:** Ensure no group is disproportionately misclassified as "negative" when they should be "positive."

**Why It’s Important**

* **Fairness Impact:** High FNR for a group means they’re unfairly denied benefits (e.g., loans, job screenings).
* **Real-world Harm:** E.g., a hiring AI rejecting qualified women more often than men.
* **Legal/Regulatory Risk:** Bias can violate anti-discrimination laws (e.g., EU AI Act, U.S. Equal Credit Opportunity Act).

**How a Model Might Fail This Metric**

1. **Skewed Training Data** – Underrepresentation of a group leads to poor predictions for them.
2. **Biased Features** – Using proxies for race/gender (e.g., ZIP codes affecting loan approvals).
3. **Threshold Imbalance** – Default decision thresholds may disadvantage certain groups.