**EMAIL DETECTION**

import numpy as np

emails = [

"Get free money now!",

"Important meeting tomorrow.",

"Great offers on our products.",

"Limited time offer, buy now!",

"Meeting agenda attached.",

"Claim your prize!",

"Don't miss out on this deal."

]

labels = [

1, # Spam

0, # Not spam

1, # Spam

1, # Spam

0, # Not spam

1, # Spam

0 # Not spam

]

vocab = set(word.lower() for email in emails for word in email.split())

word\_to\_index = {word: i for i, word in enumerate(vocab)}

X = np.zeros((len(emails), len(vocab)))

for i, email in enumerate(emails):

for word in email.split():

X[i, word\_to\_index[word.lower()]] += 1

X /= X.sum(axis=1, keepdims=True)

y = np.zeros((len(labels), 2))

for i, label in enumerate(labels):

y[i, label] = 1

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return sigmoid(x) \* (1 - sigmoid(x))

class ANN:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

self.weights\_input\_hidden = np.random.randn(input\_size, hidden\_size)

self.biases\_input\_hidden = np.zeros(hidden\_size)

self.weights\_hidden\_output = np.random.randn(hidden\_size, output\_size)

self.biases\_hidden\_output = np.zeros(output\_size)

def forward(self, X):

self.hidden\_input = np.dot(X, self.weights\_input\_hidden) + self.biases\_input\_hidden

self.hidden\_output = sigmoid(self.hidden\_input)

self.output = sigmoid(np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.biases\_hidden\_output)

return self.output

def train(self, X, y, learning\_rate=0.1, num\_epochs=1000):

for epoch in range(num\_epochs):

self.forward(X)

output\_error = y - self.output

output\_delta = output\_error \* sigmoid\_derivative(self.output)

hidden\_error = np.dot(output\_delta, self.weights\_hidden\_output.T)

hidden\_delta = hidden\_error \* sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += learning\_rate \* np.dot(self.hidden\_output.T, output\_delta)

self.biases\_hidden\_output += learning\_rate \* np.sum(output\_delta, axis=0)

self.weights\_input\_hidden += learning\_rate \* np.dot(X.T, hidden\_delta)

self.biases\_input\_hidden += learning\_rate \* np.sum(hidden\_delta, axis=0)

# Calculate and print the loss

loss = np.mean(np.abs(output\_error))

if epoch % 100 == 0:

print(f'Epoch {epoch}: Loss = {loss}')

input\_size = X.shape[1]

hidden\_size = 64

output\_size = 2 # Binary classification

model = ANN(input\_size, hidden\_size, output\_size)

model.train(X, y)

predictions = model.forward(X)

binary\_predictions = np.argmax(predictions, axis=1)

accuracy = np.mean(binary\_predictions == labels)

print(f'Accuracy: {accuracy}')