**HW 6 Report: Neural Networks**

**A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.NET 1**

**A graph with blue and orange dots

AI-generated content may be incorrect.A graph with blue and orange dots

AI-generated content may be incorrect.**For NET1, we can see that classifying Versicolor vs Virginica using sepal features results in a much lower accuracy compared to classifying Setosa vs Virginica using petal features. The hidden layer cannot learn the complexity of the relationship between Versicolor vs Virginica.

From these plots, it is clear the Setosa vs Virginica data points can be linearly separated, while Versicolor vs Virginica cannot. This is the core reason that the accuracy in the first set is much lower. To solve this, we could increase the number of layers so that the model may learn any non-linear relationships.

**A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.NET 2**

Increasing the number of neurons in a layer can allow the model to learn more complicated relationships. In set 1 increasing the number of neurons from 5 to 20 resulted in a 9% more accurate model. Because of set 2’s simple relationship, NET2 still has no trouble classifying the Setosas from Virginicas.

**A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.NET 3**

Adding a layer to NET1 resulted in a 4% increase in accuracy on set 1. A 4% increase is not substantial and could likely be to a slightly better train-test-split. This means adding 1 extra layer had little to no effect on the accuracy for set 1. Classifying Versicolors vs Virginicas using sepal features may be too complicated 2 hidden layers. It is possible that using more hidden layers, along with increasing the number of neurons per layer could result in greater accuracy; however, this could also easily lead to overfitting reducing the accuracy of the model.

**A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.NET 4**

Utilizing all 4 features enables the model to successfully classify Versicolors vs Virigincas. Up until NET4, the model had little success classifying set 1 using sepal features. This leads me to believe there is either not enough data provided or there is no way to draw a distinction between the two classes using sepal width and length. Introducing pedal features enables the model to finally find a pattern that correctly predicts the data.

**Full Code Implementation**

import numpy as np

import pandas as pd

from typing import Tuple

import matplotlib.pyplot as plt

from sklearn import datasets

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

# read\_data, get\_df\_shape, data\_split are the same as HW3

def read\_data(filename: str) -> pd.DataFrame:

d = pd.read\_csv(filename)

df = pd.DataFrame(data=d)

return df

def extract(df: pd.DataFrame, class1, class2, feat1, feat2) -> Tuple[pd.DataFrame, pd.Series]:

filtered\_df = df[df['variety'].isin([class1, class2])]

features = filtered\_df[[feat1, feat2]]

labels = filtered\_df['variety'].map({class1: 0, class2: 1})

return features, labels

def extract\_all\_feat(df: pd.DataFrame, class1, class2) -> Tuple[pd.DataFrame, pd.Series]:

filtered\_df = df[df['variety'].isin([class1, class2])]

features = filtered\_df.drop('variety', axis=1)

labels = filtered\_df['variety'].map({class1: 0, class2: 1})

return features, labels

def sigmoid(z):

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

def function\_derivative(z):

fd = sigmoid(z)

return fd \* (1 - fd)

class NN:

def \_\_init\_\_(self, features, hidden\_layers, output\_neurons, learning\_rate):

self.features = features

self.hidden\_layers = hidden\_layers

self.output\_neurons = output\_neurons

self.learning\_rate = learning\_rate

# initialize weights

self.W = []

self.b = []

layer\_sizes = [features] + hidden\_layers + [output\_neurons]

for i in range(len(layer\_sizes) - 1):

self.W.append(np.random.randn(layer\_sizes[i], layer\_sizes[i+1]))

self.b.append(np.zeros(layer\_sizes[i+1]))

def train(self, X, t, epochs=1000):

costs = []

for epoch in range(epochs):

# forward pass

nets, activations = self.forwardPass(X)

# backpropagation

self.backpropagate(t, nets, activations)

# find the cost function

if epoch % 10 == 0:

loss = np.square(np.subtract(t, activations[-1])).mean()

costs.append(loss)

return costs

def forwardPass(self, X):

X = np.array(X)

nets = []

activations = [X]

a = X

for i in range(len(self.W)):

z = a.dot(self.W[i]) + self.b[i]

a = sigmoid(z)

nets.append(z)

activations.append(a)

return nets, activations

def backpropagate(self, t, nets, activations):

t = np.array(t)

deltas = [None] \* len(self.W)

deltas[-1] = (activations[-1] - t) \* function\_derivative(nets[-1])

for i in reversed(range(len(self.W) - 1)):

deltas[i] = function\_derivative(nets[i]) \* deltas[i+1].dot(self.W[i+1].T)

for i in range(len(self.W)):

d\_W = activations[i].T.dot(deltas[i])

d\_b = np.sum(deltas[i], axis=0)

self.W[i] -= self.learning\_rate \* d\_W

self.b[i] -= self.learning\_rate \* d\_b

def predict(self, X):

a = X

for i in range(len(self.W)):

z = a.dot(self.W[i]) + self.b[i]

a = sigmoid(z)

return (a > 0.5).astype(int)

def NET(set, test, hidden\_layers, title=""):

nn = NN(features=set[0].shape[1], hidden\_layers=hidden\_layers, output\_neurons=1, learning\_rate=0.01)

cost = nn.train(set[0], set[1])

y\_pred = nn.predict(test[0])

acc = accuracy(test[1], y\_pred)

print(f"{title} - Accuracy: {acc:.2f}")

plt.plot(cost)

plt.title(f"{title} - Accuracy: {acc:.2f}")

plt.ylabel("Loss")

plt.xlabel("Epochs")

plt.show()

def visualizeData(X, y, title, classes):

X = X.values

y = y.flatten()

for label, class\_name in zip(np.unique(y), classes):

idx = y == label

plt.scatter(X[idx, 0], X[idx, 1], label=class\_name)

plt.title(title)

plt.ylabel('Width')

plt.xlabel('Length')

plt.legend(loc="upper right")

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

def accuracy(t, y\_pred):

accuracy = np.sum(np.array(t) == np.array(y\_pred)) / len(t)

return accuracy

train\_df = read\_data("./iris\_training\_data.csv")

test\_df = read\_data("./iris\_testing\_data.csv")

X1, t1 = extract(train\_df, 'Versicolor', 'Virginica', 'sepal.length', 'sepal.width')

X1\_test, t1\_test = extract(test\_df, 'Versicolor', 'Virginica', 'sepal.length', 'sepal.width')

X2, t2 = extract(train\_df, 'Setosa', 'Virginica', 'petal.length', 'petal.width')

X2\_test, t2\_test = extract(test\_df, 'Setosa', 'Virginica', 'petal.length', 'petal.width')

t1 = t1.values.reshape([len(t1),1])

t2 = t2.values.reshape([len(t2),1])

t1\_test = t1\_test.values.reshape([len(t1\_test),1])

t2\_test = t2\_test.values.reshape([len(t2\_test),1])

set1 = (X1, t1)

set2 = (X2, t2)

set1\_test = (X1\_test, t1\_test)

set2\_test = (X2\_test, t2\_test)

X3, t3 = extract\_all\_feat(train\_df, 'Versicolor', 'Virginica')

X3\_test, t3\_test = extract\_all\_feat(test\_df, 'Versicolor', 'Virginica')

X4, t4 = extract\_all\_feat(train\_df, 'Setosa', 'Virginica')

X4\_test, t4\_test = extract\_all\_feat(test\_df, 'Setosa', 'Virginica')

t3 = t3.values.reshape([-1, 1])

t4 = t4.values.reshape([-1, 1])

t3\_test = t3\_test.values.reshape([-1, 1])

t4\_test = t4\_test.values.reshape([-1, 1])

set3 = (X3, t3)

set4 = (X4, t4)

set3\_test = (X3\_test, t3\_test)

set4\_test = (X4\_test, t4\_test)

visualizeData(X1, t1, "Versicolor vs Virginica (sepal features)", ("Versicolor", "Virginica"))

visualizeData(X2, t2, "Setosa vs Virginica (petal features)", ("Setosa", "Virginica"))

# NET 1

NET(set1, set1\_test, hidden\_layers=[5], title="NET1: Versicolor vs Virginica (sepal features)")

NET(set2, set2\_test, hidden\_layers=[5], title="NET1: Setosa vs Virginica (petal features)")

# NET 2

NET(set1, set1\_test, hidden\_layers=[20], title="NET2: Versicolor vs Virginica (sepal features)")

NET(set2, set2\_test, hidden\_layers=[20], title="NET2: Setosa vs Virginica (petal features)")

# NET 3

NET(set1, set1\_test, hidden\_layers=[20], title="NET3: Versicolor vs Virginica (sepal features)")

NET(set2, set2\_test, hidden\_layers=[10, 5], title="NET3: Setosa vs Virginica (petal features)")

# NET 4

NET(set3, set3\_test, hidden\_layers=[5], title="NET4: Versicolor vs Virginica (all features)")

NET(set4, set4\_test, hidden\_layers=[5], title="NET4: Setosa vs Virginica (all features)")