Module 6 Lab

DSC 4310 - Machine Learning

Noah Choate

Part A

```
import numpy as np

#def twoSpirals(N):
#  np.random.seed(1)
#  n = np.sqrt(np.random.rand(N,1)) * 780 * (2*np.pi)/360
#  x = -np.cos(n)*n
#  y = np.sin(n)*n
#  return (np.vstack((np.hstack((x,y)),np.hstack((-x,-y)))),
#          np.hstack((np.ones(N)*-1,np.ones(N))))
#X, y = twoSpirals(300)
```

```
def sigmoid(z, grad=False):
  if grad:
    return z * (1. - z)
  return 1. / (1. + np.exp(-z))
```

```
w1 = np.array([-0.16595599, 0.44064899, -0.99977125,
               -0.39533485, -0.70648822, -0.81532281]).reshape(2, 3)

w2 = np.array([-0.62747958, -0.30887855,
               -0.20646505, 0.07763347,
               -0.16161097, 0.370439]).reshape(3, 2)

w1 = np.reshape(w1, (2, 3))
w2 = np.reshape(w2, (3, 2))
```

```
x = [7.08535569, 5.20423916]
z = np.matmul(x, w1)
print(z)
```

```
[ -3.23327433  -0.55457883 -11.32686981]
```

```
o1 = sigmoid(z)
print(o1)
```

```
[3.79325739e-02 3.64802739e-01 1.20447479e-05]
```

```
z2 = np.matmul(o1, w2)
print(z2)
```

```
[-0.09912288  0.01660881]
```

```
o2 = sigmoid(np.matmul(o1, w2))
print(o2)
```

```
[0.47523955 0.50415211]
```

Calculating Loss

```
y = [1, 0]
```

```
lambda1 = 0.01
N = 300

L = np.square(y-o2).sum()/(2*N) + lambda1 *(np.square(w1).sum()+np.square(w2).sum())/(2*N)
print(L)
```

```
0.0009366140675459939
```

The Backward Pass

```
dL_dz2 = (o2 - y) / N * sigmoid(o2, grad=True)

# Compute gradient w.r.t. w2
dL_dw2 = np.outer(o1, dL_dz2) + (lambda1 / N) * w2

# Compute gradient w.r.t. hidden layer
dL_do1 = np.dot(w2, dL_dz2)
dL_dz1 = dL_do1 * sigmoid(o1, grad=True)

# Compute gradient w.r.t. w1
dL_dw1 = np.outer(x, dL_dz1) + (lambda1 / N) * w1

print("Gradient of w1:\n", dL_dw1)
print("Gradient of w2:\n", dL_dw2)
```

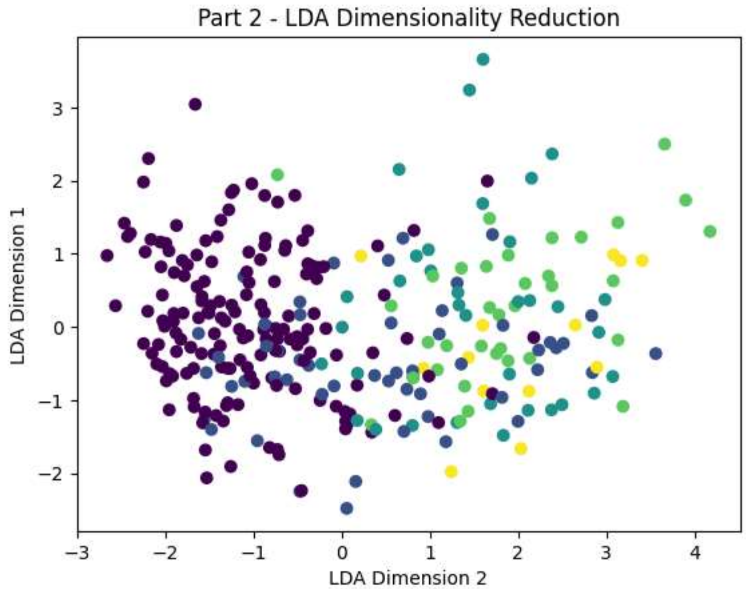
```
Gradient of w1:
 [[ 3.16932665e-05  2.16107231e-04 -3.33064112e-05]
 [ 1.41642703e-05  1.24393889e-04 -2.71632531e-05]]
Gradient of w2:
 [[-3.74632359e-05  5.63943853e-06]
 [-1.66019328e-04  1.55840604e-04]
 [-5.39228659e-06  1.23530266e-05]]
```

Part B

```
# Load Libraries/Packages and Data for Part B
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import balanced_accuracy_score

!wget https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data
df = pd.read_csv('processed.cleveland.data', header=None)
```

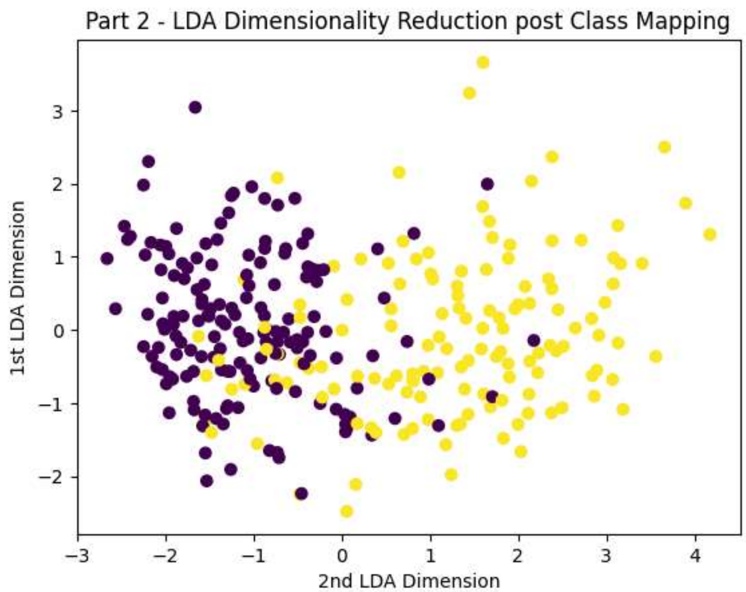
```
--2025-02-22 23:28:42--  https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘processed.cleveland.data.2’

processed.cleveland     [ <=>                ]  18.03K  --.-KB/s    in 0.01s

2025-02-22 23:28:43 (1.42 MB/s) - ‘processed.cleveland.data.2’ saved [18461]
```

```
# Prepare data
df = df.apply(pd.to_numeric, errors='coerce').dropna()

X = df[[0,1,2,3,4,5,6,7,8,9,10,11,12]].values
# Target vals
y = df[13].values

# Excercise LDA to 2 components
dr = LinearDiscriminantAnalysis(n_components=2)
X_ = dr.fit_transform(X, y)


# Plot values
import matplotlib.pyplot as plt
plt.title('Part 2 - LDA Dimensionality Reduction')
plt.xlabel('LDA Dimension 2')
plt.ylabel('LDA Dimension 1')
plt.scatter(X_[:,0], X_[:,1], c=y, cmap='viridis')
plt.show()
```




```
df[13].replace(to_replace=[1, 2, 3, 4], value=1, inplace=True)
df[13].replace(to_replace=[0], value=-1, inplace=True)
print(df.head())

plt.title("Part 2 - LDA Dimensionality Reduction post Class Mapping")
plt.ylabel("1st LDA Dimension")
plt.xlabel("2nd LDA Dimension")
plt.scatter(X_[:,0], X_[:,1], c=y)
plt.show()
```

```
<ipython-input-37-724cc0c9cc66>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[13].replace(to_replace=[1, 2, 3, 4], value=1, inplace=True)
<ipython-input-37-724cc0c9cc66>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[13].replace(to_replace=[0], value=-1, inplace=True)
      0    1    2      3      4    5    6      7    8    9   10   11   12  13
0  63.0  1.0  1.0  145.0  233.0  1.0  2.0  150.0  0.0  2.3  3.0  0.0  6.0  -1
1  67.0  1.0  4.0  160.0  286.0  0.0  2.0  108.0  1.0  1.5  2.0  3.0  3.0   1
2  67.0  1.0  4.0  120.0  229.0  0.0  2.0  129.0  1.0  2.6  2.0  2.0  7.0   1
3  37.0  1.0  3.0  130.0  250.0  0.0  0.0  187.0  0.0  3.5  3.0  0.0  3.0  -1
4  41.0  0.0  2.0  130.0  204.0  0.0  2.0  172.0  0.0  1.4  1.0  0.0  3.0  -1
```



3-I

```
# Splits data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.2, random_state=7)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize weights + parameters
np.random.seed(7)
input_dim = X_train.shape[1]
hidden_dim = 4  # Number of hidden neurons in diagram
output_dim = 1  # Specifies binary classification in target variable after remapping

w1 = np.random.randn(input_dim, hidden_dim) * 0.01
w2 = np.random.randn(hidden_dim, output_dim) * 0.01

# Define hyperparameters per instructions
lambda_ = 0.00001
alpha = 0.001
t = 100000

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))


for i in range(t):
```

```
        # Forward pass
        z1 = np.dot(X_train, w1)
        o1 = sigmoid(z1)
        z2 = np.dot(o1, w2)
        o2 = sigmoid(z2)

        # Loss computation
        loss = np.square(y_train.reshape(-1, 1) - o2).sum() / (2 * len(y_train)) + lambda_ * (np.square(w1).sum() + np.square(w2).sum()) / (2 *

        # Backprop
        dL_dz2 = (o2 - y_train.reshape(-1, 1)) * o2 * (1 - o2)
        dL_dw2 = np.dot(o1.T, dL_dz2) + lambda_ * w2 / len(y_train)

        dL_dz1 = np.dot(dL_dz2, w2.T) * o1 * (1 - o1)
        dL_dw1 = np.dot(X_train.T, dL_dz1) + lambda_ * w1 / len(y_train)

        # Gradient descent for updating weights
        w1 -= alpha * dL_dw1
        w2 -= alpha * dL_dw2

        # Print loss every 5k iterations
        if i % 5000 == 0:
            print(f"Iteration {i} - Loss = {loss}")

# Testing Data
z1_test = np.dot(X_test, w1)
o1_test = sigmoid(z1_test)
z2_test = np.dot(o1_test, w2)
o2_test = sigmoid(z2_test)

y_pred = np.where(o2_test >= 0.5, 1, -1)

# Balanced accuracy values
balanced_acc = balanced_accuracy_score(y_test, y_pred)
print(f"Balanced Accuracy: {balanced_acc:.2f}")
```

```
Iteration 0 - Loss = 0.66567390909376
Iteration 5000 - Loss = 0.39123610142900217
Iteration 10000 - Loss = 0.3881157231042612
Iteration 15000 - Loss = 0.38695190428845394
Iteration 20000 - Loss = 0.3863244166884061
Iteration 25000 - Loss = 0.38591947753762
Iteration 30000 - Loss = 0.385627542455359
Iteration 35000 - Loss = 0.3854008396511476
Iteration 40000 - Loss = 0.3852155266850328
Iteration 45000 - Loss = 0.38505852977052896
Iteration 50000 - Loss = 0.38492213344893594
Iteration 55000 - Loss = 0.3848014902957918
Iteration 60000 - Loss = 0.3846933790172935
Iteration 65000 - Loss = 0.38459554741698726
Iteration 70000 - Loss = 0.38450634826224783
Iteration 75000 - Loss = 0.38442452923224996
Iteration 80000 - Loss = 0.3843491070979833
Iteration 85000 - Loss = 0.38427928949233686
Iteration 90000 - Loss = 0.38421442441427683
Iteration 95000 - Loss = 0.38415396640427646
Balanced Accuracy: 0.50
```

** Changed numbers back to instructions and ran again **

After experimenting with some of the hyperparameter values, I could not necessarily find values that resulted in a significantly improved model. For example, after raising the alpha values, the loss values marginally improved by almost .003. While this is a quantifiable improvement, I would not consider it significant. Another notable aspect that I found throughout various runs is that the loss values seemed to stabilize around the 10000-15000 iteration marks therefore it may be overkill to be doing 95000 iterations. This is most likely due to the dataset not being linearly seperable.

Part C

```
# Load necessary packages and redefine x and y back to original values
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam
```

```
import keras_tuner as kt
import numpy as np

def twoSpirals(N):
  np.random.seed(1)
  n = np.sqrt(np.random.rand(N,1)) * 780 * (2*np.pi)/360
  x = -np.cos(n)*n
  y = np.sin(n)*n
  return (np.vstack((np.hstack((x,y)),np.hstack((-x,-y)))),
          np.hstack((np.ones(N)*-1,np.ones(N))))
X, y = twoSpirals(300)


def build_model(hp):
    model = Sequential()

    # Input layer
    model.add(Dense(16, activation=hp.Choice('activation', ['relu', 'sigmoid']), input_shape=(2,)))

    # Hidden layers (per Instructions)
    num_layers = hp.Choice('num_layers', [2, 3, 4, 8])

    for _ in range(num_layers - 1):  # Subtract 1 because first layer is already added
        model.add(Dense(1, activation=hp.Choice('activation', ['relu', 'sigmoid'])))

    # Output layer
    model.add(Dense(1, activation='softmax'))

    # Compile the model
    model.compile(
        optimizer=SGD(learning_rate=0.01),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    return model

# Define the tuner (Grid Search)
tuner = kt.GridSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    directory='mlp_hyperparam_search',
    project_name='mlp_tuning'
)

# Perform the search for the best hyperparameters
tuner.search(X, y, epochs=50, validation_split=0.2, batch_size=60, verbose=1)

# Retrieve optimal hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"Best number of layers: {best_hps.get('num_layers')}")
print(f"Best activation function: {best_hps.get('activation')}")

# Train the best model with optimal hyperparameters
best_model = tuner.hypermodel.build(best_hps)
best_model.fit(X, y, epochs=100, batch_size=60, validation_split=0.2, verbose=1)
```

```
Reloading Tuner from mlp_hyperparam_search/mlp_tuning/tuner0.json
Best number of layers: 3
Best activation function: relu
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/ops/nn.py:907: UserWarning: You are using a softmax over axis -1 of a tensor of shap
  warnings.warn(
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.4034 - loss: 0.6830 - val_accuracy: 1.0000 - val_loss: 0.7233
Epoch 2/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3829 - loss: 0.6395 - val_accuracy: 1.0000 - val_loss: 0.7538
Epoch 3/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3806 - loss: 0.5979 - val_accuracy: 1.0000 - val_loss: 0.7844
Epoch 4/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3759 - loss: 0.5570 - val_accuracy: 1.0000 - val_loss: 0.8153
Epoch 5/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3916 - loss: 0.5251 - val_accuracy: 1.0000 - val_loss: 0.8463
Epoch 6/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3902 - loss: 0.4891 - val_accuracy: 1.0000 - val_loss: 0.8774
Epoch 7/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3845 - loss: 0.4510 - val_accuracy: 1.0000 - val_loss: 0.9086
```

```
Epoch 8/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.3674 - loss: 0.4035 - val_accuracy: 1.0000 - val_loss: 0.9399
Epoch 9/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3906 - loss: 0.3909 - val_accuracy: 1.0000 - val_loss: 0.9712
Epoch 10/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3785 - loss: 0.3479 - val_accuracy: 1.0000 - val_loss: 1.0024
Epoch 11/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.3767 - loss: 0.3160 - val_accuracy: 1.0000 - val_loss: 1.0337
Epoch 12/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.3907 - loss: 0.3029 - val_accuracy: 1.0000 - val_loss: 1.0650
Epoch 13/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.3778 - loss: 0.2587 - val_accuracy: 1.0000 - val_loss: 1.0962
Epoch 14/100
```