LAB Manual

**Experiment No. 4**

PART B

|  |  |
| --- | --- |
| Roll No: C009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 12/01/24 | Date of Submission : 12/01/24 |
| Grade : |  |

**B.1 Software Code written by student:**

# C009

# Samarth Borade

# BTI SEM 10

#EXP 4: Reducing the bias and variance of a neural network

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

df=pd.read\_csv('diabetes.csv')

df.head()

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']

X = df[columns]

y = df['Outcome']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = Sequential()

model.add(Dense(units=8, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dense(units=8, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=100)

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy \* 100:.2f}%')

#Plot

plt.plot(history.history['accuracy'], label='Accuracy')

plt.title('Testing Accuracy vs. Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

#EPOCHS =500

model = Sequential()

model.add(Dense(units=8, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=500, batch\_size=32)

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy \* 100:.2f}%')

plt.plot(history.history['accuracy'], label='Accuracy')

plt.title('Testing Accuracy vs. Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

#EPOCHS =1000

model = Sequential()

model.add(Dense(units=8, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dense(units=8, activation='sigmoid'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=1000, batch\_size=32)

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy \* 100:.2f}%')

plt.plot(history.history['accuracy'], label='Accuracy')

plt.title('Testing Accuracy vs. Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

#EPOCHS =2000

model = Sequential()

model.add(Dense(units=8, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=8, activation='relu'))

model.add(Dense(units=8, activation='sigmoid'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=2000, batch\_size=100)

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss:.4f}')

print(f'Test Accuracy: {accuracy \* 100:.2f}%')

plt.plot(history.history['accuracy'], label='Accuracy')

plt.title('Testing Accuracy vs. Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

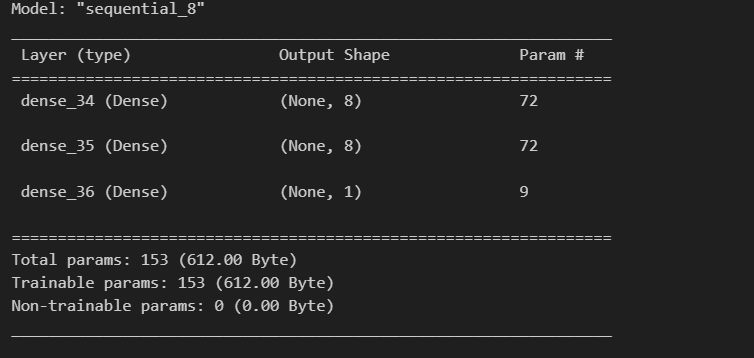
**B.2 Output:**

**Output:**

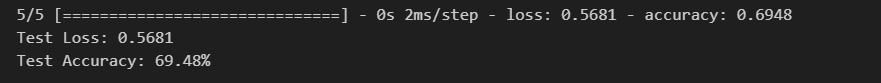
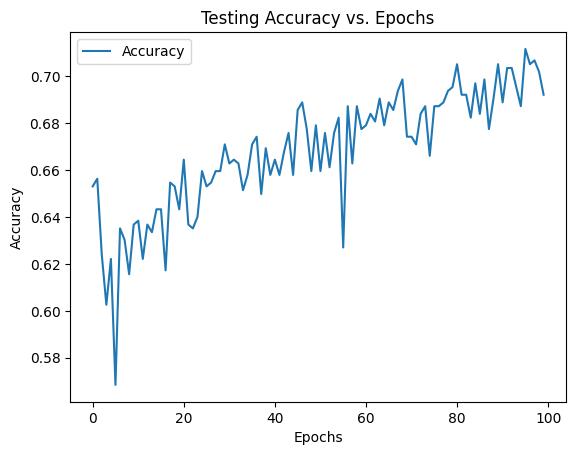
**Hidden Layers: 1**

**Epochs = 100**

Activation function- relu, sigmoid, relu, sigmoid

****

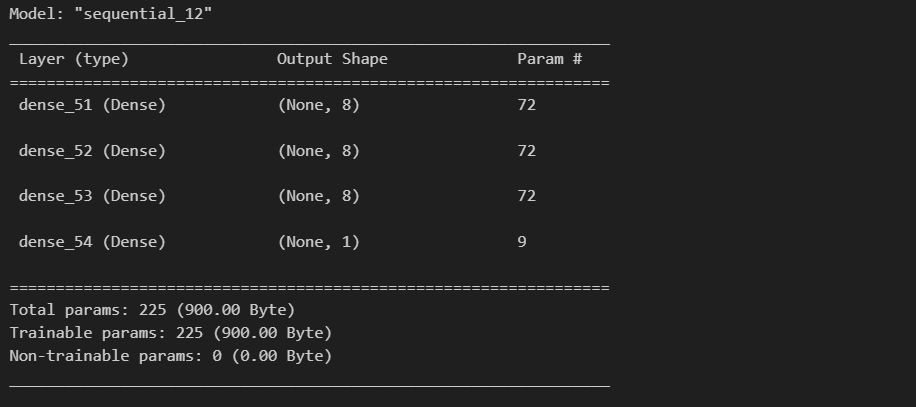
Loss and Accuracy:

**** ****

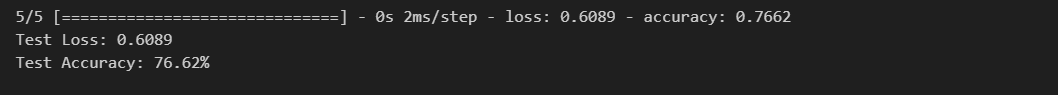
**Hidden Layers: 2**

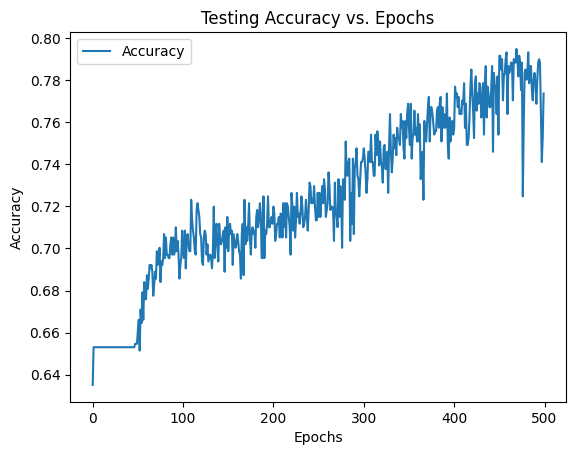
**Epochs = 500**

Activation function- relu, sigmoid, relu, sigmoid



Loss and Accuracy:

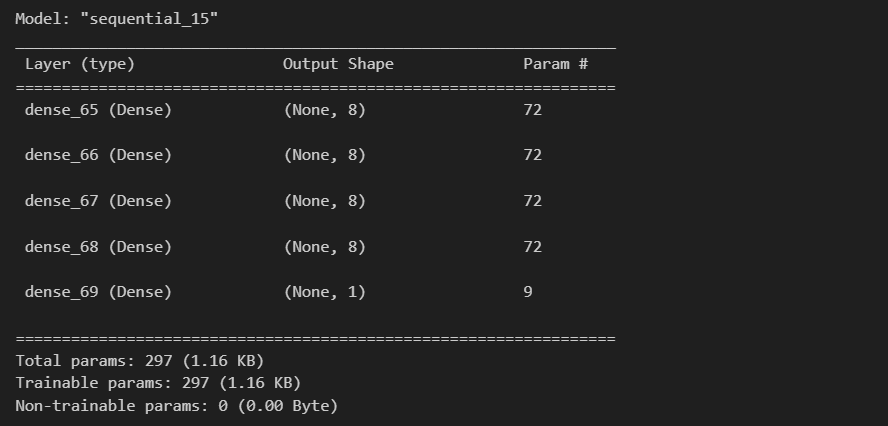




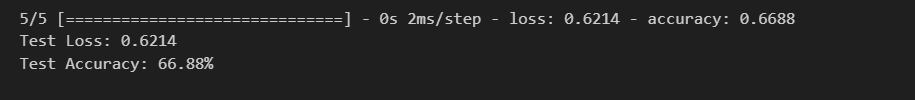
**Epochs= 1000**

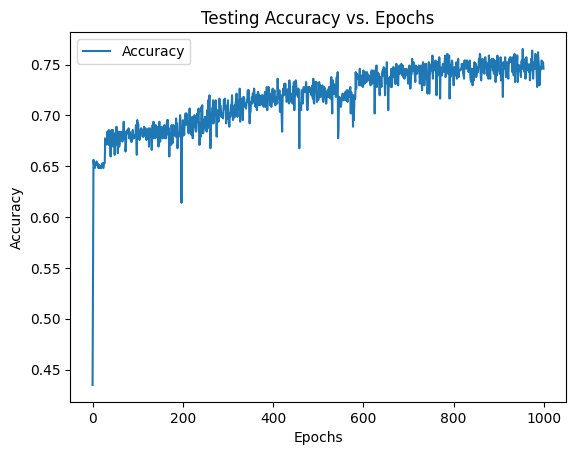
**Hidden layers= 3**

**Activation function= relu, relu, sigmoid, relu, sigmoid**



Loss and Accuracy:

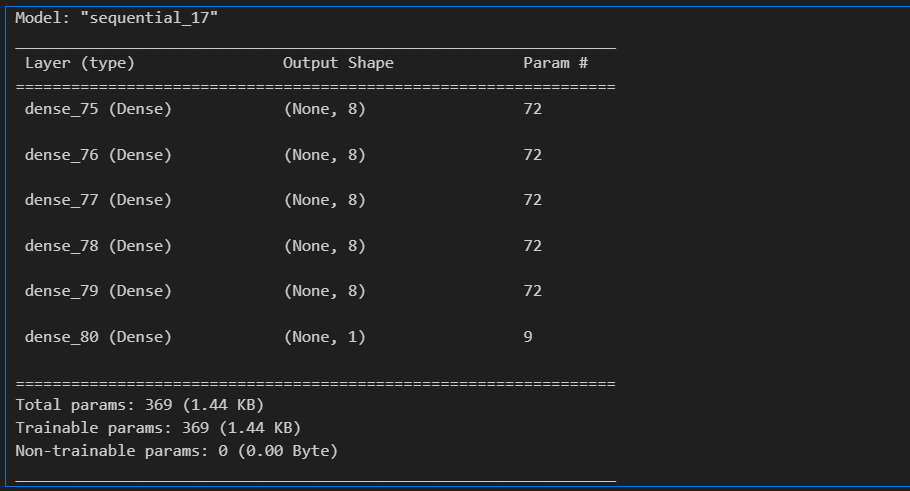




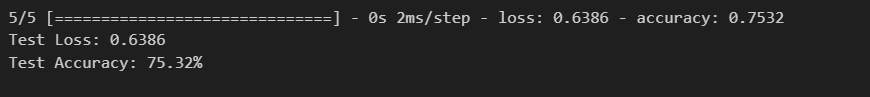
**Epochs=2000**

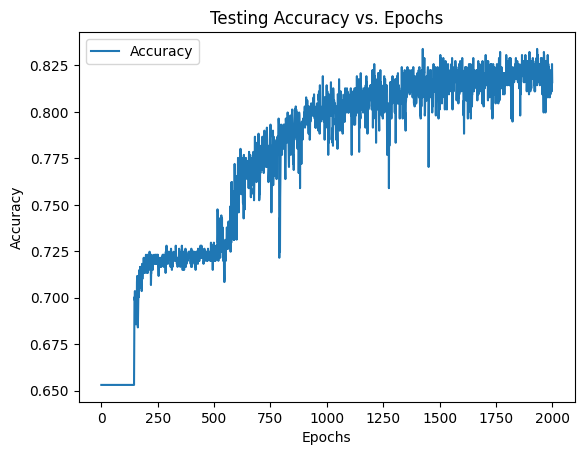
**Hidden layers= 4**

**Activation function= relu, relu, relu, sigmoid, sigmoid, sigmoid**



Loss and Accuracy:





**B.3 Observations and learning:**

After testing different setups, the model with four hidden layers using ReLU and a final sigmoid layer performed the best. It was trained for 2000 rounds and achieved around 75.32% accuracy in predicting diabetes, with a loss of 0.6386. While it's reasonably effective, tweaking settings and trying more data could make it better. Keeping an eye on how it handles larger datasets and exploring techniques like regularization might also help make it more reliable.

**B.4 Conclusion:**   
The neural network models were trained on the diabetes dataset with varying architectures and epochs. The best-performing model had four hidden layers of eight units each, achieving a test accuracy of approximately 80%. However, increasing the model complexity or epochs did not significantly improve performance, suggesting an optimal balance between model capacity and training duration.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*