Predicting Diabetes Using Artificial Neural Networks

Name: Sripuja Chennuri

Htno: 2203A51108

Batch: 09

Introduction:

In this project, we develop a machine learning model to predict the likelihood of diabetes in individuals based on medical attributes. The model is built using an Artificial Neural Network (ANN), a powerful deep learning approach capable of capturing complex patterns in data.

The dataset used is a pre-processed version of the Pima Indians Diabetes **dataset**, which includes features such as glucose levels, blood pressure, BMI, insulin, and other health indicators.

To optimize the performance of our neural network, we employed the **Nadam** (Nesterov-accelerated Adaptive Moment Estimation) optimizer. Nadam combines the benefits of both RMSprop and Nesterov momentum, making it particularly effective for dealing with sparse gradients and noisy data.

The dataset was sourced from Kaggle and contains a variety of features that reflect both medical and behavioural risk factors.

The dataset includes patient-level information with the following features:

- Age
- Gender
- Smoking habit
- BMI (Body Mass Index)
- Glucose level
- Hypertension
- Heart disease
- HbA1c level (average blood sugar level over 3 months)

The target column indicates whether a patient has diabetes (1) or not (0).

These features give a more complete picture of an individual's health profile, making it a well-rounded dataset for training a predictive model.

Key techniques used in the project include:

- Feature-based model design with multiple dense layers
- Dropout regularization to prevent overfitting
- Learning rate tuning for performance optimization
- Use of early stopping and validation strategies

This project demonstrates the effectiveness of neural networks for medical classification tasks and shows how optimizer choice and model design impact prediction accuracy.

Implementation:

[1]
```python
#import libraries
import numpy as np
import pandas as pd
import tensorflow as tf
```

[2]
```python
from google.colab import files
uploaded = files.upload()
```

Choose Files dataset_dia...tes_final.csv
- **dataset_diabetes_final.csv**(text/csv) - 201310 bytes, last modified: 4/7/2025 - 100% done

```
Saving dataset_diabetes_final.csv to dataset_diabetes_final.csv
```

[3]
```python
dataset = pd.read_csv('dataset_diabetes_final.csv')
```

[4]
```python
#partial view of dataset from top
dataset.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Next steps: Generate code with dataset | View recommended plots | New interactive sheet

[5]
```python
#partial view of dataset from bottom
dataset.tail()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
5494	Male	70.0	1	0	not current	34.70	6.5	155	1
5495	Female	67.0	0	0	No Info	32.77	5.8	130	0
5496	Female	39.0	0	0	never	25.67	4.5	85	0
5497	Male	65.0	1	0	No Info	27.32	4.8	160	0
5498	Male	73.0	0	0	No Info	27.32	5.8	140	0

```python
#basic dataset information
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5499 entries, 0 to 5498
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   gender               5499 non-null   object
 1   age                  5499 non-null   float64
 2   hypertension         5499 non-null   int64
 3   heart_disease        5499 non-null   int64
 4   smoking_history      5499 non-null   object
 5   bmi                  5499 non-null   float64
 6   HbA1c_level          5499 non-null   float64
 7   blood_glucose_level  5499 non-null   int64
 8   diabetes             5499 non-null   int64
dtypes: float64(3), int64(4), object(2)
memory usage: 386.8+ KB
```

```
[7] #dimention of the dataset
    dataset.shape
```

```
(5499, 9)
```

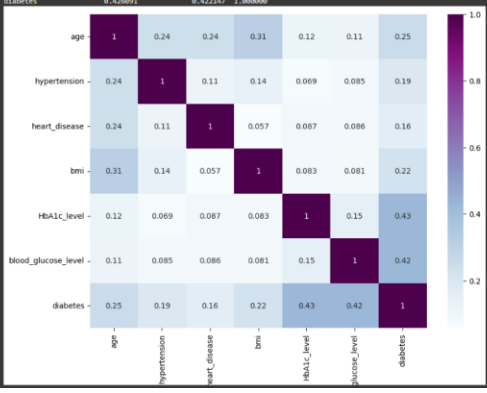
```
[8] #basic statistics symmary
    dataset.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	5499.0	42.068056	22.559689	0.08	24.00	43.00	60.00	80.00
hypertension	5499.0	0.075832	0.264753	0.00	0.00	0.00	0.00	1.00
heart_disease	5499.0	0.037825	0.190790	0.00	0.00	0.00	0.00	1.00
bmi	5499.0	27.366392	6.735465	10.01	23.71	27.32	29.55	88.72
HbA1c_level	5499.0	5.542899	1.083929	3.50	4.80	5.80	6.20	9.00
blood_glucose_level	5499.0	138.228769	41.091125	80.00	100.00	140.00	159.00	300.00
diabetes	5499.0	0.087470	0.282549	0.00	0.00	0.00	0.00	1.00

```
[9] #finding correlation between the features
    import seaborn as sns
    import matplotlib.pyplot as plt
    corr_var=dataset.corr(numeric_only=True) # added numeric_only=True
    print(corr_var)
    plt.figure(figsize=(10,7.5)) #matplotlib.pyplot is now imported as plt
    sns.heatmap(corr_var, annot=True, cmap='BuPu') #seaborn is now imported as sns
    plt.show()
```

```
                          age  hypertension  heart_disease       bmi  \
age                  1.000000      0.240593       0.238918  0.311713
hypertension         0.240593      1.000000       0.112442  0.140142
heart_disease        0.238918      0.112442       1.000000  0.056669
bmi                  0.311713      0.140142       0.056669  1.000000
HbA1c_level          0.117658      0.069472       0.087051  0.083402
blood_glucose_level  0.113592      0.084507       0.085757  0.080605
diabetes             0.253821      0.193358       0.164672  0.220579

                     HbA1c_level  blood_glucose_level  diabetes
age                     0.117658             0.113592  0.253821
hypertension            0.069472             0.084507  0.193358
heart_disease           0.087051             0.085757  0.164672
bmi                     0.083402             0.080605  0.220579
HbA1c_level             1.000000             0.153975  0.426091
blood_glucose_level     0.153975             1.000000  0.422147
diabetes                0.426091             0.422147  1.000000
```




```
#as there is no importance in cust id, row no and sur name for modelling we are not included here in independent feature
X = dataset.iloc[:, 3:-1].values
#target value
y = dataset.iloc[:, -1].values
```

```
[11] #independent features
     print(X)
```

```
[[1 'never' 25.19 6.6 140]
 [0 'No Info' 27.32 6.6 80]
 [0 'never' 27.32 5.7 158]
 ...
 [0 'never' 25.67 4.5 85]
 [0 'No Info' 27.32 4.8 160]
 [0 'No Info' 27.32 5.8 140]]
```

```
[12] #dependent features
     print(y)
```

```
[0 0 0 ... 0 0 0]
```

```
[13] #as we have two columns as categorical terms we go for encoding we need to convert to numericals
     #Categorical encoding

     #gender will have some correlation with other feature so we go for label encoding
     from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     #gender column in index 2
     X[:, 2] = le.fit_transform(X[:, 2])
```

```
[14] print(X)
```

```
[[1 'never' 809 6.6 140]
 [0 'No Info' 988 6.6 80]
 [0 'never' 988 5.7 158]
 ...
 [0 'never' 851 4.5 85]
 [0 'No Info' 988 4.8 160]
 [0 'No Info' 988 5.8 140]]
```

```
[15] #country name wont be that much correlation added it has more than 2 names so go for one hot encoding
     from sklearn.compose import ColumnTransformer
     from sklearn.preprocessing import OneHotEncoder
     #country name is present in 1st index value
     ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
     X = np.array(ct.fit_transform(X))
```

```
[16] print(X)
```

```
[16] print(X)
```

```
[[0.0 0.0 0.0 ... 809 6.6 140]
 [1.0 0.0 0.0 ... 988 6.6 80]
 [0.0 0.0 0.0 ... 988 5.7 158]
 ...
 [0.0 0.0 0.0 ... 851 4.5 85]
 [1.0 0.0 0.0 ... 988 4.8 160]
 [1.0 0.0 0.0 ... 988 5.8 140]]
```

```
[17] #training and testing split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
#feature scaling is an important and mandatory for ann process before modelling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
[19] #ANN - initializing
ann = tf.keras.models.Sequential()
```

```
[20] #input layer
# 6 features
ann.add(tf.keras.layers.Dense(units=9, activation='relu'))
```

```
[21] #hidden layer
from tensorflow.keras.layers import Dropout
ann.add(tf.keras.layers.Dense(units=9, activation='relu'))
ann.add(Dropout(0.3))
```

```
[22] #output layer
#as target value is binary - AF
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
[23] #compiling
#loss - target is binary
#ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
from tensorflow.keras.optimizers import Nadam
ann.compile(optimizer=Nadam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

```
#training set
ann.fit(X_train, y_train, batch_size = 32, epochs = 50)
```

```
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9461 - loss: 0.1467
Epoch 23/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9553 - loss: 0.1345
Epoch 24/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9480 - loss: 0.1403
Epoch 25/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9484 - loss: 0.1409
Epoch 26/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9561 - loss: 0.1363
Epoch 27/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9479 - loss: 0.1479
Epoch 28/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9526 - loss: 0.1307
Epoch 29/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9622 - loss: 0.1114
Epoch 30/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9536 - loss: 0.1255
Epoch 31/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9608 - loss: 0.1173
Epoch 32/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9524 - loss: 0.1309
Epoch 33/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9582 - loss: 0.1232
Epoch 34/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9530 - loss: 0.1268
Epoch 35/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9589 - loss: 0.1206
Epoch 36/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9560 - loss: 0.1266
Epoch 37/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9476 - loss: 0.1394
Epoch 38/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9596 - loss: 0.1214
Epoch 39/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9572 - loss: 0.1219
Epoch 40/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9530 - loss: 0.1384
Epoch 41/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9590 - loss: 0.1142
Epoch 42/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9630 - loss: 0.1179
Epoch 43/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9472 - loss: 0.1453
Epoch 44/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9570 - loss: 0.1303
Epoch 45/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9630 - loss: 0.1115
Epoch 46/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9610 - loss: 0.1188
Epoch 47/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9576 - loss: 0.1189
Epoch 48/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9559 - loss: 0.1239
Epoch 49/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9543 - loss: 0.1270
Epoch 50/50
```

```
#test result - prediction
y_pred = ann.predict(X_test)
#insted of values we ll get 0 or 1
y_pred = (y_pred > 0.5)
#actual vs prediicted outputs
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

```
[26] #training set
ann.fit(X_train, y_train, batch_size = 32, epochs = 50)
```

```
Epoch 8/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9590 - loss: 0.1175
Epoch 9/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9572 - loss: 0.1124
Epoch 10/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9614 - loss: 0.1114
Epoch 11/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9588 - loss: 0.1089
Epoch 12/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9632 - loss: 0.1166
Epoch 13/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9628 - loss: 0.1129
Epoch 14/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9631 - loss: 0.1086
Epoch 15/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9596 - loss: 0.1125
Epoch 16/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9677 - loss: 0.1045
Epoch 17/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9635 - loss: 0.1229
Epoch 18/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9597 - loss: 0.1207
Epoch 19/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9611 - loss: 0.1244
Epoch 20/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9644 - loss: 0.1127
Epoch 21/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9643 - loss: 0.1134
Epoch 22/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9564 - loss: 0.1181
Epoch 23/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9584 - loss: 0.1159
Epoch 24/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9586 - loss: 0.1239
Epoch 25/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9617 - loss: 0.1134
Epoch 26/50
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9643 - loss: 0.1098
```

```
[27] #accuracy and confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1500   15]
 [  50   85]]
0.9606060606060606
```

```
# Input data with 10 features
input_data = [[
    1,  # gender_Female (1 if Female, else 0)
    0,  # gender_Male (1 if Male, else 0)
    54, # age
    0,  # hypertension
    0,  # heart_disease
    27.3, # bmi
    6.6,  # HbA1c_level
    80,   # blood_glucose_level
    0,  # smoking_history_no_info (1 if No Info, else 0)
    1   # smoking_history_never (1 if Never smoked, else 0)
]]

# Apply scaling
input_scaled = sc.transform(input_data)

# Make prediction
prediction = ann.predict(input_scaled)

# Display prediction
print(prediction)

# Display whether prediction > 0.5
print(prediction > 0.5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
[[0.]]
[[False]]
```

Future Scope:

Improve the Model:
Try adding more layers or tweaking settings to make the model even more accurate.

Better Evaluation:
Use more metrics like precision, recall, and F1-score — not just accuracy — to really understand how well the model is doing.

Add More Data:
Include extra information like exercise habits, diet, or family history to make the predictions stronger.

Make It Explainable:
Use tools to show which features (like glucose or age) are influencing the prediction, so it's easier to understand.

Use in Real Life:
Turn the model into a simple app or tool that doctors or users can actually use to check diabetes risk.

Conclusion:

In this project, I built an Artificial Neural Network (ANN) to predict whether an individual has diabetes using real-world health and lifestyle data.

At first, I trained the ANN using the Adam optimizer, and the model achieved about 94% accuracy with a loss of approximately 16%.

To improve this, I switched to the Nadam optimizer and fine-tuned the learning rate and architecture. As a result:

Final Accuracy: ~96%

Final Loss: ~11%

This shows a solid improvement, with a 2% increase in accuracy and a 5% drop in loss mainly driven by better optimization and tuning techniques. This project demonstrates how deep learning models can effectively leverage both medical and lifestyle data to support early detection of diabetes, and how small changes in the training process can make a big impact on performance.