HW 2 Diabetes Classification Using ANN

1. Objective:

In this assignment, you will implement a simple feedforward artificial neural network (ANN) using NumPy only, trained with Stochastic Gradient Descent (SGD) and square error loss. The goal is to classify whether a patient has diabetes based on medical attributes.

You will build the ANN from scratch — no use of ML libraries like TensorFlow, Keras, or PyTorch is allowed.

Goal:

- Understand how a neural network works internally (forward + backward pass)
- Learn how to preprocess real-world medical data
- Apply your model to a real binary classification problem

2. Dataset: Pima Indians Diabetes Database

Download: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/data

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Source: The dataset comes from the UCI Machine Learning Repository and is centered around Pima Indians, particularly women who may be at risk for diabetes.
- Goal: The objective is to predict whether a patient has diabetes or not based on various medical indicators (e.g., glucose level, blood pressure, etc.). This is a binary classification problem where the target variable is 0 (no diabetes) or 1 (has diabetes).
- Data Size: The dataset contains 768 records, each representing a patient with 8 features and a label indicating whether or not they have diabetes, as Table 1.

Table 1

Feature	Type	Description
Pregnancies	int	Number of times pregnant
Glucose	int	Plasma glucose concentration
BloodPressure	int	Diastolic blood pressure (mm Hg)
SkinThickness	int	Triceps skinfold thickness (mm)
Insulin	int	2-Hour serum insulin (mu U/ml)
BMI	float	Body mass index
DiabetesPedigreeFunction	float	Diabetes pedigree function
Age	int	Age (years)
Outcome	int (0/1)	Class label (1 = diabetic, 0 = non-diabetic)

3. Data Preprocessing

The dataset requires some preprocessing:

- **Missing Values**: Some features, such as SkinThickness and Insulin, may have missing values, which need to be handled (e.g., by filling in with the median or mean).
- **Normalization**: Since the features have varying ranges, normalization or standardization is necessary for training stable models (e.g., scaling to a 0-1 range or using z-scores).
- **Categorical Features**: The only categorical feature here is the target variable (Outcome), which is binary. There are no other categorical features that require encoding.

4. Steps and Coding Tasks:

You must implement all components of the neural network from scratch using NumPy only. No use of scikit-learn, TensorFlow, Keras, PyTorch, etc.

Step 1: Data Preprocessing

- Load CSV data

```
!kaggle datasets download -d uciml/pima-indians-diabetes-database
!unzip pima-indians-diabetes-database.zip
```

```
Dataset URL: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
License(s): CC0-1.0
Archive:  pima-indians-diabetes-database.zip
  inflating: diabetes.csv
```

- Normalize each feature (recommended: min-max or standardization)
- Split data into training (80%) and testing (20%)
- Shuffle the data before training.

Step 2: Build Neural Network

Implement a simple fully connected neural network with:

- **Input layer**: 8 neurons (one per feature)
- **1 hidden layer**: configurable neurons (suggested: 8 or 16)
- **Activation function**: sigmoid or ReLU for hidden layer
- **Output layer**: 1 neuron with **sigmoid** activation
- **Loss function**:

$$L = \frac{1}{2}(\hat{y} - y)^2$$

- **Training method**: **SGD**, one sample at a time

Step 3: Training Loop

- **Perform forward pass**
- **Compute square error loss**
- **Compute gradients via backpropagation**
- **Update weights using SGD:**

$$w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w}$$

- Train for multiple **epochs** (e.g., 1000+), plotting loss vs. epoch

Step 4: Evaluation

- Use test set to compute accuracy

- Print final weights, confusion matrix, and test accuracy

5. Submission:

Please zip and submit the following to iLearning:

- A Python script or Jupyter notebook containing the code: `ann_diabetes.py` or `ann_diabetes.ipynb`
- A summary report of the results in PDF format `report.pdf`, including
  - Your model structure and hyperparameters
  - Plot of loss over epochs
  - Final test set accuracy
  - Explanation of your results, observations, or challenges