Experiment 2

Aim: *To implement and evaluate a simple neural network for binary classification on the make_moons dataset. We will compare the performance of the neural network with and without a hidden layer. This experiment focuses on using a neural network architecture with one hidden layer (containing 5 neurons) and applying the ReLU activation function for the hidden layer and the Sigmoid function for the output layer.*

Theory:

1. **Neural Networks**: A neural network consists of layers of neurons. Each neuron receives input, processes it, and produces output based on a predefined mathematical function. A feedforward neural network consists of an input layer, one or more hidden layers, and an output layer.
2. **Activation Functions**:
    a. **ReLU (Rectified Linear Unit)**: A commonly used activation function in hidden layers. It replaces all negative values with zero and passes all positive values as is. It helps in learning non-linear relationships.
    b. **Sigmoid**: A function that outputs values between 0 and 1, typically used in the output layer for binary classification tasks. It is effective for predicting probabilities.
3. **Backpropagation**: A process used for training neural networks. It calculates the error at the output, then propagates the error backward through the network to update the weights using gradient descent.
4. **Binary Cross-Entropy Loss**: A loss function used for binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels.

5. **Dataset - make_moons**:

a. A synthetic 2D dataset often used for testing machine learning models.
b. It consists of two interlocking crescent-shaped classes, making it challenging for linear models to classify.

Methodology:

1. **Dataset Preparation**: We used the make_moons function from the sklearn.datasets module to generate a 2D dataset with 200 samples and a noise level of 0.1. The dataset was then split into training and test sets (80% for training and 20% for testing).
2. **Preprocessing**: The features were standardized using StandardScaler to ensure that all input features have a mean of 0 and a standard deviation of 1.
3. **Neural Network Architecture**:
    a. **Input Layer**: 2 neurons (corresponding to the 2 features in the dataset).
    b. **Hidden Layer**: 5 neurons with ReLU activation.
    c. **Output Layer**: 1 neuron with Sigmoid activation to output probabilities for binary classification.
4. **Training**:
    a. We used **gradient descent** to optimize the weights. The loss was computed using binary cross-entropy, and the gradients were propagated backward using backpropagation.
    b. The network was trained for 1000 epochs, with the weights updated after each epoch using the computed gradients.
5. **Evaluation**: The network's performance was evaluated on the test set by comparing the predicted labels to the actual labels. The accuracy was computed as the proportion of correct predictions.
6. **Visualization**: The decision boundary was plotted for both the training and test sets to visually assess how well the model learned the non-linear boundaries in the data.

Results:

- **Model with No Hidden Layer**:
    - The neural network without a hidden layer performed relatively poorly. The accuracy on the test set was lower compared to the model with a hidden layer. The decision boundary was linear, which struggled to separate the crescent-shaped classes in the make_moons dataset.
- **Model with Hidden Layer**:
    - The network with one hidden layer (5 neurons) performed significantly better. The accuracy on the test set was higher, and the decision boundary showed a more complex shape, successfully separating the two classes. The non-linear decision boundary indicated that the hidden layer allowed the model to learn more complex patterns in the data.
- **Test Accuracy**:
    - **Without Hidden Layer**: ~0.75 (may vary slightly depending on training)
    - **With Hidden Layer**: ~0.90 (may vary slightly depending on training)
- **Decision Boundary**:
    - The decision boundary without a hidden layer was linear, while the decision boundary with a hidden layer was non-linear and more closely matched the underlying structure of the data.

Discussion:

- **Without a Hidden Layer**: The neural network failed to capture the non-linear nature of the make_moons dataset. The decision boundary was linear and thus could not separate the two crescent-shaped classes effectively.
- **With a Hidden Layer**: The introduction of a hidden layer with ReLU activation allowed the model to capture the complex relationships between the input

features. The non-linear decision boundary enabled the network to separate the two classes with much higher accuracy.

- **Activation Function Choice**: Using ReLU in the hidden layer allowed for better learning of non-linearities, while the Sigmoid function in the output layer provided an appropriate probability-based classification for binary outcomes.
- **Overfitting**: The model with a hidden layer performed well, but there's always a risk of overfitting with more complex models, especially if the dataset is small or the network is overly large.

Conclusion:

1. **Hidden Layer Benefits**: The experiment clearly demonstrates that adding a hidden layer to the neural network improves its ability to classify non-linearly separable data. The model with a hidden layer achieved higher accuracy compared to the model without a hidden layer.
2. **ReLU Activation**: The use of ReLU in the hidden layer effectively enabled the network to capture the non-linear structure of the data. Without ReLU, the network would have struggled to differentiate between the classes.
3. **Generalization**: For more complex datasets with non-linear relationships between features, incorporating hidden layers is crucial for improving model performance.
4. **Future Work**:
    a. **Hyperparameter Tuning**: Further experiments with different numbers of hidden neurons, learning rates, and training epochs could yield even better results.
    b. **Regularization**: To avoid overfitting, regularization techniques such as dropout or L2 regularization could be explored.
    c. **Advanced Architectures**: More complex architectures, such as deeper networks with multiple hidden layers, could be tested for even more complex datasets.