Assignment: Implementing and Comparing Gradient Descent and ID3

Intro to Machine Learning

Objective

Submitted by	
Nicole Frumkin	211615372
Alona Gertskin	207787540
Nitzan Monfred	316056126

The goal of this assignment is to:

1. **Understand and implement a single-layer neural network using gradient descent.**
2. **Understand and implement the ID3 algorithm for decision trees.**
3. **Compare the performance of these algorithms on the same dataset.**

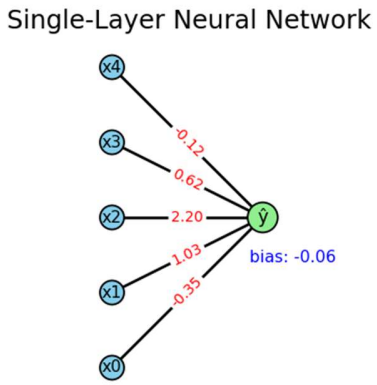
Instructions

Part 1: Single-Layer Neural Network with Gradient Descent

1. **Implementation:**
   - **Implement a single-layer neural network with sigmoid activation in Python.**
   - **Implement and use gradient descent to minimize the error with cross-entropy loss.**
   - **Compute the gradients (no use of libraries like PyTorch or TensorFlow).**
   - **Initialize weights and biases randomly. Think about different distributions such as normal and uniform. You are encouraged to read online about common initializations.**
   - **Try different learning rates and weights initializations.**
   - **Train with Gradient descent until a stopping condition (of your choice) is reached. Examples for stopping conditions: goal accuracy, reaching convergence, fixed number of epochs.**
   - **Output:**
   - **In the final report include the results for the best learning rate and initialization, mention some other options that you've tried and briefly explain why you chose the ones that you did (in what sense they were the best).**
   - **Plot the loss curves (train and test) over epochs.**
   - **Briefly analyse the plots, do they match the theory?**
   - **Provide the final weights and biases.**
   - **Explain your choice of stopping condition.**

We wrote a code that creates a single layer neural network.

Here is an example output for 5 features:



We noticed that the breast cancer dataset contained 30 features, so that's what we used to train our data.

For the weights, we initialized them to be uniformly distributed between $-\frac{1}{\sqrt{30}}$ and $\frac{1}{\sqrt{30}}$, where 30 is the number of features. This is similar to the Xavier initialization method which helped to balance the starting values of the weights. We tried to initialize the weights using different distributions, like normal distribution, but we found that uniform distribution gave the best results.

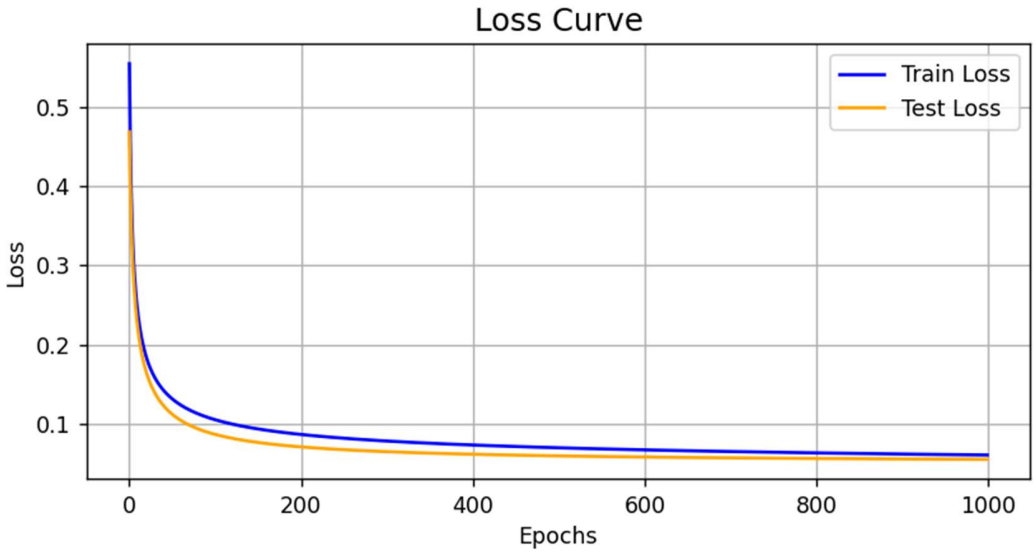
We chose a learning rate of 0.01 because higher rates than that made the model unstable and caused it to diverge. But smaller values made the training very slow or even fail to converge.

We chose to stop the training after 1000 epochs because around this value the train loss and the test loss started to get very close to each other. Therefore, we didn't use the condition for early stopping because there were no signs of overfitting.

The final weights and bias were:

```
Final Weights:
 [-0.563 -0.739 -0.539 -0.619 -0.179  0.206 -0.519 -0.65  -0.006  0.264
 -0.925  0.022 -0.554 -0.682 -0.19   0.643  0.018 -0.186  0.277  0.597
 -0.948 -0.941 -0.721 -0.652 -0.72  -0.199 -0.768 -0.87  -0.788 -0.173]
Final Bias: 0.492
Final Accuracy: 0.991
```

Loss curves: we can see in the plot below that both the training and testing loss decrease over time and come close together in the end. Meaning the model was learning in a stable way and generalizing well.



Part 2: ID3 Algorithm

1. **Implementation:**
   - **Implement the ID3 algorithm for constructing a decision tree.**
   - **Use information gain with entropy to determine the best feature to split the dataset at each node.**
   - **Allow the algorithm to handle categorical and numerical data.**
2. **Output:**
   - **Visualize the constructed tree (using any suitable Python library or by printing the tree structure).**

We implemented the ID3 algorithm from to create a decision tree classifier.

We used entropy-based information gain for feature selection at each node. The tree recursively splits data until reaching pure nodes or maximum depth.

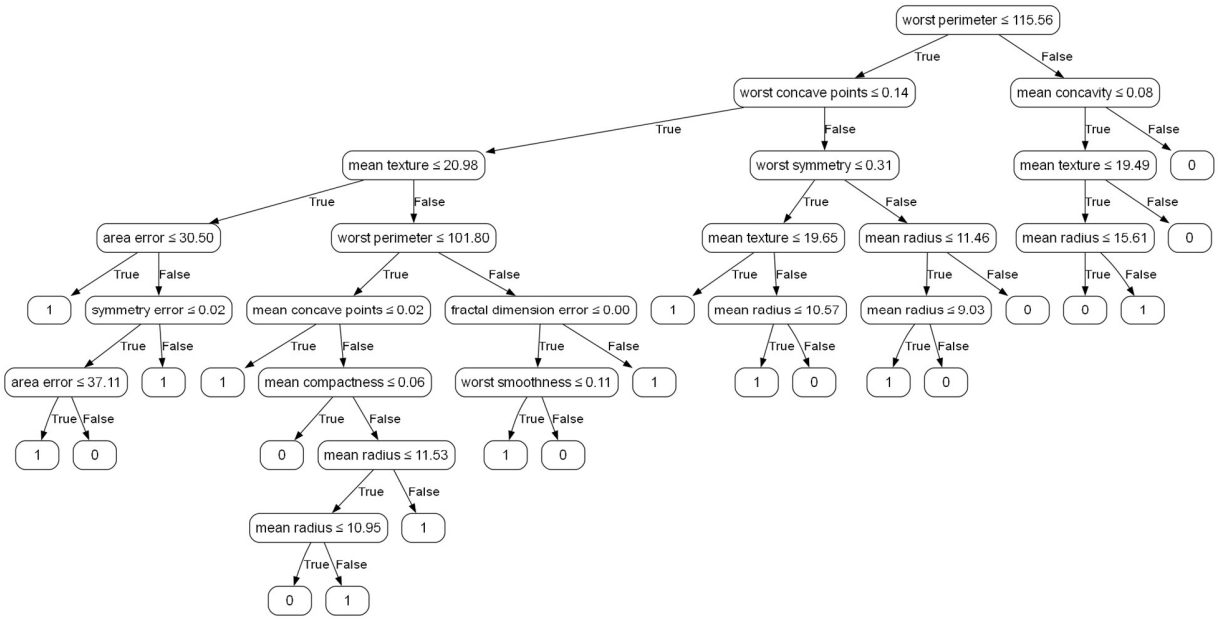
Since the breast cancer dataset has 30 continuous features, we adapted the algorithm to use threshold-based splits (feature ≤ threshold) rather than categorical equality.

The maximum depth was set to 8 levels - the minimum depth required to achieve 100% training accuracy.

Results:

- Training Accuracy: 100.00%
- Test Accuracy: 92.98%

Final Tree Structure:



Part 3: Comparison

1. **Dataset:**
   - **Use Wisconsin breast cancer dataset from sklearn https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_ca ncer.html**
2. **Tasks:**
   - **Load the dataset and split it to 80% train and 20% test.**
   - **Train both the neural network and the decision tree on the dataset.**
   - **Compare the accuracy (percent of the correct predictions out of all of the predictions) models.**
   - **Discuss the strengths and weaknesses of each algorithm based on your findings.**
3. **Submission:**
   - **Include:**
     - **Python code for both implementations.**
     - **A report discussing your methodology, results, and comparison.**
     - **Plots and visualizations to support your findings.**

Methodology

We compared two machine learning models for binary classification on the Breast Cancer Wisconsin dataset:

1. A simple feedforward neural network with a single layer and sigmoid activation.
2. A manually implemented ID3 decision tree algorithm with a controlled maximum depth.

The dataset was preprocessed using StandardScaler, and split into 80% training and 20% testing. The neural network was trained using gradient descent to minimize cross-entropy loss. The decision tree used entropy and information gain for splits, and a maximum depth of 8 to avoid overfitting.

Results

After training:

- Neural Network Accuracy:
  - Train: ~99.0%
  - Test: ~97.0%
- Decision Tree Accuracy:
  - Train: 98.24%
  - Test: 96.49%

The neural network slightly outperformed the decision tree on both training and test data, showing better generalization while being slightly more sensitive to optimization dynamics (e.g., learning rate, number of epochs).

Comparison

- Neural Network:
  - Pros: Higher accuracy, adaptable via hyperparameter tuning.
  - Cons: Requires more training time and sensitive to scaling and parameter choice.
- Decision Tree:
  - Pros: Very interpretable, quick to train, non-parametric.
  - Cons: Slightly lower accuracy, can overfit without pruning or depth control.

Plot and Visualization