DEEP LEARNING

CSL 4020

Assignment **2** Report

Perceptron and FFN from Scratch

By - PRANAV PANT

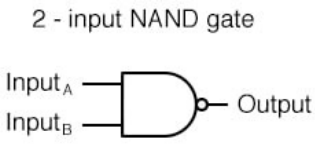
B21CS088

Perceptron and FFN

1. Design and implement a perceptron to simulate a 2-input NAND gate and 2-input XOR gate. Compute the weights.

In this question, we implemented a perceptron to simulate a 2-input NAND gate and 2-input XOR gate. The perceptron is a simple neural network model consisting of a single layer that processes inputs and generates a binary output based on a weighted sum of the inputs. The algorithm is taken from the slides provided.

(i) NAND gate

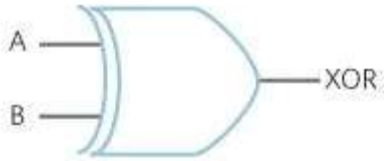


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate input – outputs

(ii) XOR gate

$$X = A \oplus B$$



A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate input outputs

Code Explanation:

- **Perceptron Class:**

- **Initialization**: We initialized the weights to zero, including a bias term.
- **Prediction:** The *predict* method computes the weighted sum of the inputs and applies a threshold to generate a binary output.
- **Training**: The *train* method iteratively updates the weights using a simple update rule whenever the model makes an incorrect prediction. The input data is preprocessed by negating all zero-class examples to ensure proper convergence.
- **Testing**: The *test* method evaluates the trained model on new input data to assess its performance.

Training Process:

The training process involves iteratively adjusting the weights until the perceptron correctly classifies all input examples or reaches the maximum number of iterations. If the model converges before the maximum iterations, the training process stops.

Results:

(i) For NAND gate

The perceptron was successfully trained to simulate a 2-input NAND gate, achieving convergence after **8 iterations** and correct classification of the gate's logic.

The final weights are: **[-3,-2]** for w1,w2 and **[-4]** for bias**.**

```
Testing the trained perceptron:
Inputs:
[[ 0  0 -1]
 [ 0  1 -1]
 [ 1  0 -1]
 [ 1  1 -1]]
Predicted outputs: [1, 1, 1, 0]
Final weights: [-3. -2. -4.]
```

Final Model Output

(ii) For XOR gate

```
Iteration 50: Updating weights
  Before update: weights = [ 1.  1. -1.]
  After update: weights = [0. 0. 0.]
Maximum iterations reached without convergence

Testing the trained perceptron:
Inputs:
[[ 0  0 -1]
 [ 0  1 -1]
 [ 1  0 -1]
 [ 1  1 -1]]
Predicted outputs: [0, 0, 0, 0]
Final weights: [0. 0. 0.]
```

From the results we can see that the **model did not converge.**

Despite the simplicity of the perceptron model, we observe that it was unable to solve the XOR problem effectively due to its linear nature. This behavior was evident as the weights failed to converge to a solution that could correctly classify all input-output pairs of the XOR gate.

Conclusion:

The iterative weight updates demonstrated that **a single-layer perceptron struggles with non-linearly separable data**, such as XOR. This highlights the necessity of using more advanced models, such as multi-layer perceptrons (MLPs) or neural networks with non-linear activation functions, to solve such problems.

2. Design a multi-layer perceptron from scratch and report your observation with accuracy.

Objective:

This project aims to classify individuals based on income using the Census Income dataset. The goal is to predict whether a person earns more than $50K or not. The classification problem is tackled using a Feed Forward Neural Network (FFN), developed from scratch, and further validated using a PyTorch implementation.

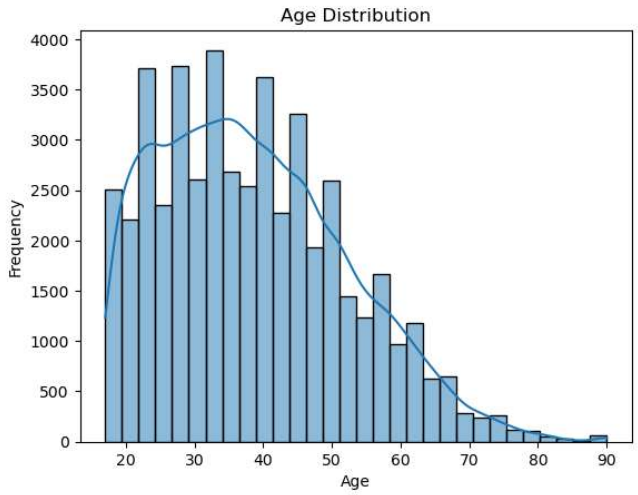
Data Preprocessing

1. **Dataset Overview**:
    - Two datasets were loaded: census_income.csv (features) and census_income_target.csv (target variable).
    - Target values were cleaned to unify the class labels (>50K, <=50K).

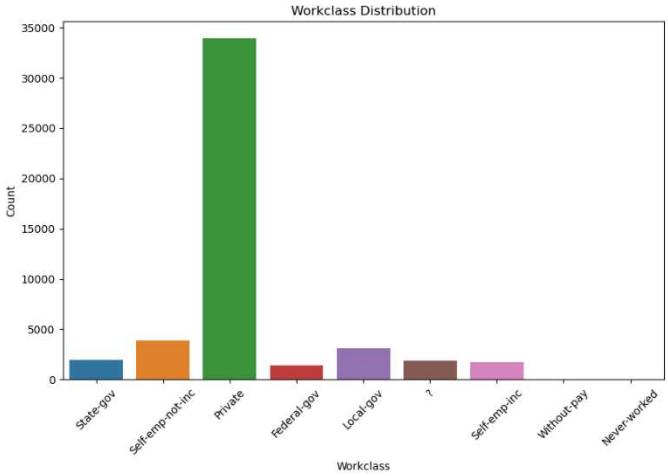
2. **Label Encoding**:
    - Categorical features like workclass, education, occupation, etc., were label encoded using LabelEncoder to convert them into numeric form.
3. **Null Values Check**:
    - There were no missing values in the dataset, ensuring a clean dataset for modeling.

Exploratory Data Analysis (EDA)

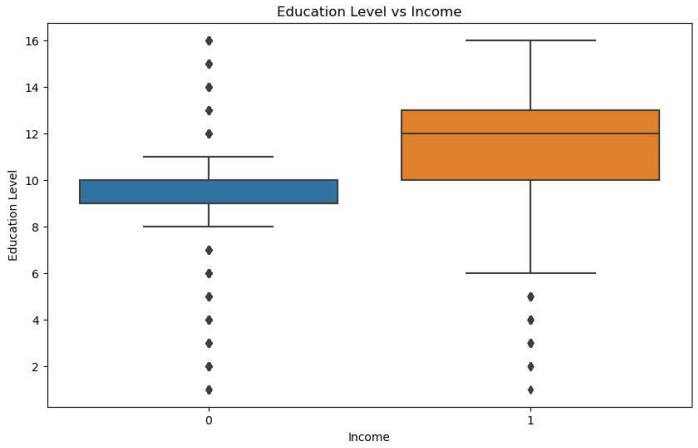
- **Age Distribution**:
    - Visualized the age distribution using a histogram to observe the frequency of individuals across various age groups. We can see that we have a continuous and spread out distribution of age in our dataset.



- **Workclass Distribution**:
    - Analyzed the distribution of the workclass column to understand the spread across different work categories. We can see that most of them work in private.



- **Education Level vs. Income**:
    - A boxplot was generated to examine the relationship between education-num and income, revealing that higher education levels generally correlate with higher income.



Feature Scaling

The dataset was scaled using **StandardScaler** to standardize the feature values, which helps improve the performance of neural networks by speeding up convergence.

Initial Neural Network (FFN) Implementation

1. **Model Architecture**:
    - **Input Layer**: Number of features.
    - **Hidden Layer**: 16 neurons.
    - **Output Layer**: 2 neurons (for binary classification).
2. **Activation Functions**:
    - **ReLU** for the hidden layer.
    - **Softmax** for the output layer to handle classification.
3. **Training Process:**

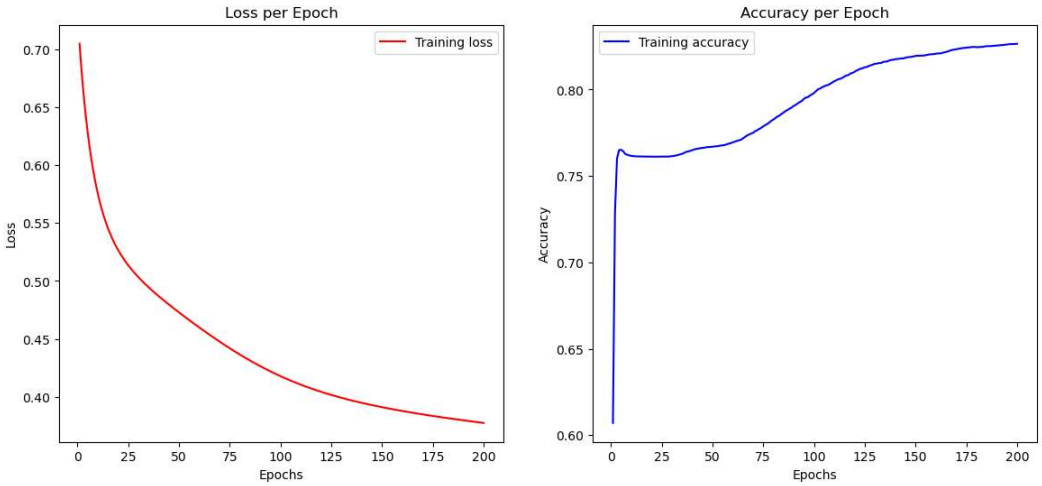
- **Loss Function:** Cross-entropy loss was used to measure the difference between the predicted and actual labels.
- **Backward Propagation:** The network's weights were updated using gradient descent, with the gradients calculated using backpropagation.
- **Training Epochs:** The model was trained for 200 epochs, with the training loss and accuracy recorded at each epoch.

4. **Results**:

   - **Overall Test Accuracy**: The model achieved good accuracy of **82%**; however, it was observed that *performance was poorer for class 1* (>50K) due **to fewer samples.**

```
Overall Test Accuracy: 0.8246

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7414
           1       0.71      0.47      0.56      2355

    accuracy                           0.82      9769
   macro avg       0.78      0.70      0.73      9769
weighted avg       0.81      0.82      0.81      9769

Class 0 Accuracy: 93.81%
Class 1 Accuracy: 46.75%
```

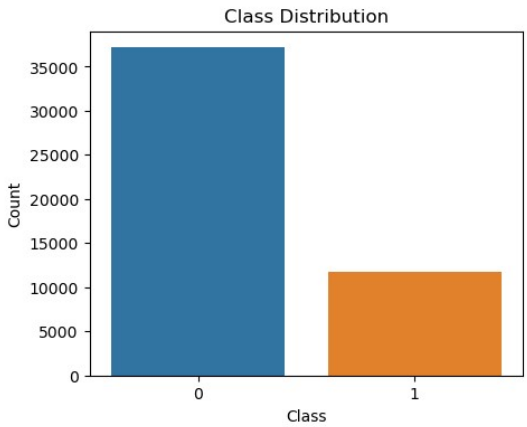
   - **Class-wise Accuracy**:
     - **<=50K**: High accuracy – 93.81%
     - **>50K**: Lower accuracy - 46.75% due to the **class imbalance** in the dataset.



Loss per Epoch graph and Accuracy per Epoch graph.

Oversampling with SMOTE

Since we get less accuracy for class 1, we look into the data distribution and see that there is a class imbalance for class 1.

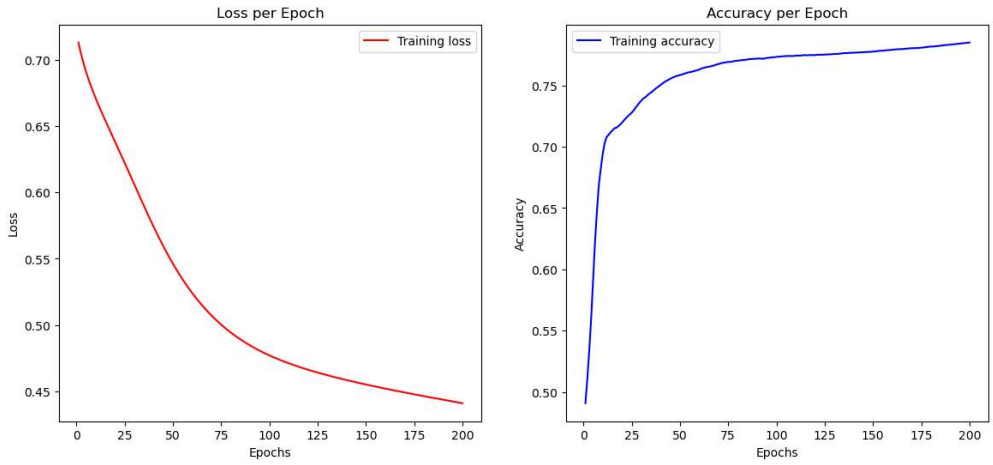


To address the class imbalance issue, **SMOTE (Synthetic Minority Over-sampling Technique)** was applied to the training data:

- After applying SMOTE, the training process was repeated using the resampled data.
- This significantly improved the accuracy for the minority class (>50K), balancing the performance across both classes.

```
Overall Test Accuracy after SMOTE: 0.7803

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.77      0.84      7414
           1       0.53      0.80      0.64      2355

    accuracy                           0.78      9769
   macro avg       0.73      0.79      0.74      9769
weighted avg       0.83      0.78      0.79      9769

Class 0 Accuracy after SMOTE: 77.45%
Class 1 Accuracy after SMOTE: 79.87%
```

  - **Overall Test Accuracy after SMOTE**: 78%.
  - **Class-wise Accuracy after SMOTE**:
    - **<=50K**: Maintained high accuracy – 77.45%
    - **>50K**: Substantial improvement – 79.9%.



Loss , Accuracy per epoch graph after SMOTE.

Hyperparameter Tuning with Grid Search

To optimize the model, a **Grid Search** was conducted to find the best parameters for:

- **Hidden Layer Size**: 8, 16, 32 neurons.
- **Learning Rate**: 0.01, 0.1, 1.
- **Number of Epochs**: 50, 100, 150.

```
Best Parameters:
{'hidden_size': 32, 'learning_rate': 1, 'num_epochs': 150}
Best Accuracy: 0.8490
```

The best parameters identified were:

- **Hidden Layer Size**: 32 neurons.
- **Learning Rate**: 0.1.
- **Epochs**: 150.

Validating Results using PyTorch:

To validate the results, the same FFN was implemented using PyTorch, a popular deep learning library. The architecture and training procedure were similar to the Numpy implementation.

- **Architecture:** The PyTorch model also had one hidden layer with and ReLU activation, followed by an output layer with binary classification.

- **Optimizer:** The model was trained using the Adam optimizer with a learning rate of 0.001.
- **Loss Function:** Cross-entropy loss was used, as in the Numpy implementation.

Training Process:

- The model was trained for 50 epochs, with the loss and accuracy tracked during training.

Results:

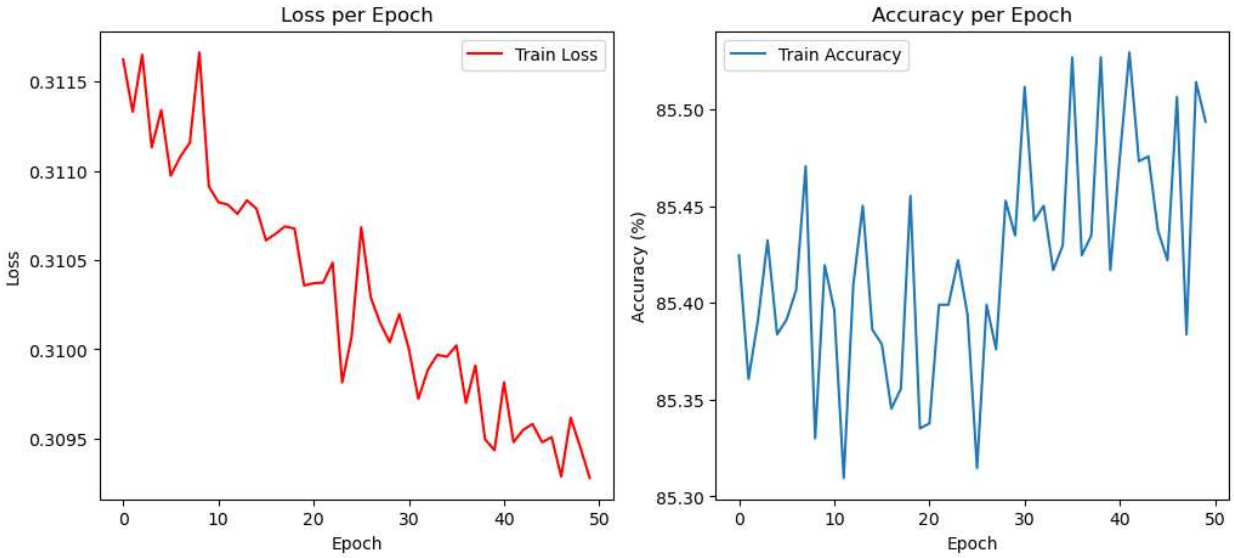
- **Test Accuracy:** 85.3%

```
Overall Test Accuracy: 0.8530

Classification Report:
              precision    recall  f1-score   support

       <=50K       0.88      0.93      0.91      7414
        >50K       0.74      0.60      0.66      2355

    accuracy                           0.85      9769
   macro avg       0.81      0.77      0.78      9769
weighted avg       0.85      0.85      0.85      9769

<=50K Accuracy: 93.32%
>50K Accuracy: 60.04%
```

Classification report and classwise accuracy for FFN using Pytorch

Graphical Analysis:

Two graphs were plotted to visualize the training process:



Loss per Epoch graph and Accuracy per Epoch graph.

Conclusion

By comparing the models:

- Both the custom-built FFN and the PyTorch model achieved similar performance (~**80**%).
- **Oversampling** using SMOTE significantly improved class-wise accuracy, especially for the minority class (>50K).
- **Grid Search** helped in finding optimal parameters, further boosting accuracy.

Overall, this project successfully demonstrated the end-to-end process of tackling an imbalanced classification problem using both a custom Feed Forward Neural Network.

----------*Report Ends*----------