Deep Neural Networks

Programming Assignment

Comparing Linear Models and
Multi-Layer Perceptrons

Course: Deep Neural Networks **Assignment**
Type: Group Programming
Maximum Marks: 10
Format: Jupyter Notebook (.ipynb)

Student Manual v1.0
November 2025

1 Assignment Overview

1.1 What You Will Do

In this assignment, you will:

1. Select a real-world dataset (≥ 500 samples, ≥ 5 features)
2. Implement a baseline linear model **from scratch** (no sklearn!)
3. Implement a Multi-Layer Perceptron (MLP) **from scratch**
4. Compare their performance on test data
5. Analyze and document your results

1.2 Learning Objectives

By completing this assignment, you will:

- Understand regression and classification from first principles
- Implement gradient descent optimization
- Build neural networks without high-level libraries
- Evaluate and compare machine learning models
- Analyze computational trade-offs in model selection

2 Dataset Requirements

2.1 Required Specifications

Requirement	**Specification**
Source	UCI ML Repository, Kaggle, or public sources
Samples	≥ 500 samples
Features	≥ 5 features (excluding target)
Data Type	Numeric and/or categorical only
Problem	Regression OR Classification

2.2 Prohibited Datasets

You CANNOT use:

- Image datasets (MNIST, CIFAR, ImageNet)
- Text datasets (sentiment analysis, NLP)
- Audio datasets (speech, music)
- Pre-processed datasets from Papers with Code
- Toy datasets with $<$ 500 samples (Iris, Wine, Diabetes)

2.3 Recommended Sources

- **UCI ML Repository:** `https://archive.ics.uci.edu/ml/`
  Examples: Breast Cancer Wisconsin, Adult Income, Abalone
- **Kaggle:** `https://www.kaggle.com/datasets`
  Filter by: Tabular data,> 500 rows
  Examples: House Prices, Credit Card Fraud
- **Other:** OpenML, Data.gov

3 Implementation Requirements

3.1 What You MUST Implement from Scratch

- For Regression:
  - Linear Regression with gradient descent
  - MLP with backpropagation
- For Binary Classification:
  - Logistic Regression with gradient descent
  - MLP with backpropagation
- For Multi-class Classification:
  - Softmax Regression with gradient descent
  - MLP with backpropagation

3.2 Allowed Libraries

You MAY use:

- `NumPy` – Array operations
- `Pandas` – Data loading and manipulation
- `Matplotlib/Seaborn` – Visualization
- `sklearn` – ONLY for:
  - `train_test_split`
  - `StandardScaler / MinMaxScaler`
  - `LabelEncoder / OneHotEncoder`

3.3 Prohibited Libraries

You CANNOT use:

- `sklearn.linear_model` (any models)
- `sklearn.neural_network` (MLPClassifier, MLPRegressor)
- `tensorflow`, `keras`, `pytorch`, `jax`
- Any high-level ML library for models

WARNING

Penalty for prohibited libraries: -5 marks

4 Assignment Structure

4.1 Dataset Selection (1 mark)

- Load and describe your dataset
- State problem type (regression/classification)
- Justify primary evaluation metric

Example Problem Statement:

"This dataset predicts tumor malignancy from 30 diagnostic measurements. I'm using ***Recall*** *as the primary metric because in medical diagnosis, false negatives (missing cancer) are more costly than false positives. Binary classification, 569 samples."*

4.2 Data Preprocessing

- Train/test split (80-20 or 90-10)
- Handle missing values if present
- Encode categorical variables
- Scale features (StandardScaler/MinMaxScaler)
- Document split ratio

4.3 Baseline Model (3 marks)

Implement from scratch with gradient descent:

```
class BaselineModel:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.lr = learning_rate
        self.n_iterations = n_iterations
        self.loss_history = []  # REQUIRED

    def fit(self, X, y):
        # Initialize weights
        # Gradient descent loop:
        #   1. Forward pass (predictions)
        #   2. Compute loss
        #   3. Compute gradients
        #   4. Update weights: w = w - lr * grad
        #   5. Store loss
        return self

    def predict(self, X):
        # Return predictions
        pass
```

Must track: `self.loss_history` with loss at each iteration!

4.4 Multi-Layer Perceptron (4 marks)

Implement MLP with:

- At least 1 hidden layer
- ReLU activation for hidden layers
- Appropriate output activation
- Forward propagation through all layers

- Backward propagation (backprop)
- Gradient descent optimization

```
class MLP:
    def __init__(self, architecture, learning_rate=0.01,
                 n_iterations=1000):
        # architecture = [input, hidden1, hidden2, ..., output]
        self.architecture = architecture
        self.loss_history = []  # REQUIRED

    def initialize_parameters(self):
        # Initialize W and b for each layer
        pass

    def forward_propagation(self, X):
        # Compute activations through all layers
        # Store in cache for backprop
        pass

    def backward_propagation(self, X, y):
        # Compute gradients using chain rule
        pass

    def fit(self, X, y):
        # Training loop with forward/backward passes
        return self

    def predict(self, X):
        # Return predictions
        pass
```

4.5 Evaluation & Comparison (2 marks)

- Calculate metrics:
  - **Regression:** MSE, RMSE, MAE, R^2
  - **Classification:** Accuracy, Precision, Recall, F1
- Create visualizations:
  - Training loss curves (baseline vs MLP)
  - Performance comparison bar chart
  - Domain-specific plots (optional)
- Write analysis (< 200 words) addressing:
  1. Which model performed better and by how much?
  2. Why did one model outperform the other?
  3. What was the computational cost difference?
  4. Any surprising findings or challenges?

5 Evaluation Criteria

5.1 Grading (10 marks total)

Component	Marks	Criteria
Dataset	1	≥ 500 samples, ≥ 5 features, valid problem
Baseline	3	Correct implementation, loss decreases, reasonable performance
MLP	4	Valid architecture, forward/backward pass, loss decreases
Comparison	2	Complete metrics, visualizations, analysis < 200 words

5.2 Deductions

- Missing `get_assignment_results()`: -2 marks
- Dataset too small: -1 mark
- Loss doesn't decrease: -1 mark per model
- Analysis ≥ 200 words: -0.5 marks
- Prohibited libraries: -5 marks

6 Submission Instructions

6.1 Complete Template

1. Download `DNN_Assignment_Template.ipynb`
2. Complete ALL sections marked `TODO`
3. Fill `get_assignment_results()` accurately
4. Run: Kernel → Restart & Run All
5. Verify no errors

6.2 Clean Notebook

Before submitting:

1. Kernel → Restart & Clear Output
2. Kernel → Restart & Run All
3. Verify all cells execute without errors
4. File → Save and Checkpoint

6.3 File Naming

Format: `YourGroupID assignment.ipynb`

6.4 Submit to LMS

1. Upload .ipynb and PDF file to assignment portal
2. Verify upload successful

7 Tips for Success

7.1 Getting Started

- **Choose simple dataset:** Binary classification easier than multi-class
- **Test with small samples:** Use 100 samples initially
- **Implement incrementally:** Get baseline working first
- **Print shapes:** Debug by printing array dimensions

7.2 Common Pitfalls to Avoid

Don't:

- Wait until last day
- Copy code without understanding
- Use sklearn models (will be caught!)
- Ignore warnings
- Forget random seeds
- Submit without running all cells

Do:

- Test frequently
- Track loss (should decrease!)
- Scale features
- Document thought process
- Use vectorized operations (NumPy)
- Comment your code

7.3 Debugging Checklist

If model not working:

1. Check data shapes at each step
2. Verify gradient computation
3. Check for NaN/Inf values
4. Try learning rates: [0.001, 0.01, 0.1]
5. Ensure loss computed correctly
6. Verify update: `w = w - lr * grad`
7. Check activation functions
8. Print intermediate values

8 Academic Integrity

8.1 Allowed

- Conceptual questions
- Clarifications about requirements
- Debugging assistance (not solutions)
- Discussion of approaches (not code sharing)

8.2 Not Allowed

- Sharing code with other students
- Using someone else's implementation
- Copying from online tutorials
- Using ChatGPT/Copilot for full solutions

9 Frequently Asked Questions

1. Q: Can I use a different dataset?
   A: Yes! Any public dataset meeting requirements is allowed.
2. Q: What if dataset has missing values?
   A: Handle them (drop rows, fill with mean/median). Document approach.
3. Q: Can I use multiple hidden layers?
   A: Yes! Minimum is 1, but you can experiment with more.
4. Q: What learning rate to use?
   A: Start with 0.01. If loss doesn't decrease, try 0.001. If explodes, try 0.1.
5. Q: Model trains too slowly?
   A: Use vectorized operations (NumPy), not loops. Reduce dataset size for testing.
6. Q: Can I submit multiple times?
   A: Check LMS settings. Usually only last submission graded.
7. Q: What if I can't finish on time?
   A: Submit what you have. Partial credit better than none.
8. Q: Will there be partial credit?
   A: Yes. Credit given for correct implementation attempts.
9. Q: Can I work with partner?
   A: No. Individual assignment. Collaboration = plagiarism.
10. Q: What if I fail verification quiz?
    A: Manual review. If legitimate work, keep score. If plagiarism, get 0.

10 Learning Outcomes

After completing this assignment, you will be able to:

1. Implement gradient descent from scratch
2. Build neural networks without frameworks
3. Understand forward and backward propagation
4. Choose appropriate evaluation metrics
5. Compare model architectures
6. Debug machine learning code
7. Present experimental results

These skills are fundamental to deep learning and will be used throughout the course!

Good luck!

Start early, test often, and don't hesitate to ask questions.

This assignment tests your understanding of neural network fundamentals.
The skills you develop here serve as foundation for advanced topics.