

DA6401 - Introduction to Deep Learning

Assignment-1 Multi-Layer Perceptron for Image Classification

General Instructions & Academic Integrity

- This is an individual assignment. Collaborations or discussions with other students are strictly prohibited.
- Tools such as ChatGPT or Claude are permitted **only** as conceptual aids or thought partners. They must not be used to generate the final code submission.
- All submissions will undergo rigorous plagiarism and AI-generated code detection. If plagiarism or unauthorized AI-assisted coding is detected, a grade of **zero** will be assigned.
- Training and test datasets must be strictly isolated. We will verify that the data split was performed properly and randomly.
- Any attempt to artificially inflate accuracy (e.g., data leakage or including test samples in the training set) will result in an immediate grade of **zero** for the entire assignment.
- You must submit both a **Public GitHub Repository** and a **Public W&B Report**. Ensure both links are accessible to the public during the evaluation phase; failure to do so will result in a negative marking penalty.
- **No extensions** will be granted beyond the provided deadline under any circumstances.
- Students are responsible for checking **Moodle** and the official course website for regular updates and clarifications regarding the assignment

Submission Details

- **Release Date:** 9th February 2026, 10:00 AM
- **Submission Deadline:** 1st March 2026, 11:59 PM
- **Gradescope:** The formal submission of your code base and W&B report must be completed via **Gradescope**.
- **Detailed Instructions:** Instructions for the Gradescope submission process will be updated and released shortly. Please monitor Moodle for the announcement.

Assignment Overview

The objective of this assignment is to build a configurable, modular Multi-Layer Perceptron (MLP) using only **NumPy**. You will implement the complete training pipeline—from forward propagation to various optimization strategies to classify the MNIST and Fashion-MNIST datasets. Your implementation must rely exclusively on **NumPy** for mathematical operations.

- **Prohibited:** PyTorch, TensorFlow, JAX, or any library providing automatic differentiation.
- **Permitted:** `keras.datasets` (for data loading), `scikit-learn` (for data splitting and confusion matrices), `matplotlib` (visualization), and `wandb` (experiment tracking).
- Project should follow this GitHub Skeleton: https://github.com/MiRL-IITM/da6401_assignment_1

1 Implementation & Evaluation Requirements (50 Marks)

1.1 Implementation Specifications

To ensure compatibility with the automated grading system, your submission must adhere to the following structural and functional requirements:

- **Project Structure:** Follow the provided GitHub skeleton exactly. Ensure all modular classes for the neural network are correctly abstracted.
- **Command Line Interface (CLI):** Your scripts (`train.py`, `inference.py`) must use `argparse` to allow full configuration. Mandatory arguments include: dataset, epochs, batch size, loss function, optimizer, learning rate, hidden layer size/count, activation function, and weight initialization method.
 1. `-d`, `--dataset`: Choose between `mnist` and `fashion_mnist`.
 2. `-e`, `--epochs`: Number of training epochs.
 3. `-b`, `--batch_size`: Mini-batch size.
 4. `-l`, `--loss`: Choice of `mean_squared_error` or `cross_entropy`.
 5. `-o`, `--optimizer`: One of `sgd`, `momentum`, `nag`, `rmsprop`, `adam`, `nadam`.
 6. `-lr`, `--learning_rate`: Initial learning rate.
 7. `-wd`, `--weight_decay`: Weight decay for L2 regularization.
 8. `-nhl`, `--num_layers`: Number of hidden layers.
 9. `-sz`, `--hidden_size`: Number of neurons in each hidden layer (list of values).
 10. `-a`, `--activation`: Choice of `sigmoid`, `tanh`, `relu` for every hidden layer.
 11. `-w_i`, `--weight_init`: Choice of `random` or `xavier`.
- **Gradient Access:** The `backward()` method in your network/layer classes must compute and store gradients. Specifically, layer objects must expose `self.grad_W` and `self.grad_b` after every call for verification.
- **Inference & Metrics:** The `inference.py` script must load serialized NumPy weights (`.npy`) and output **Accuracy, Precision, Recall, and F1-score**.
- **Model Submission:** Submit exactly one `best_model.npy` based on test F-1 score and a `best_config.json` representing your optimized weights and the model configuration.
- **Note:** It is advised to keep the number of hidden layers ≤ 6 and number of hidden neurons per layer ≤ 128 , throughout all the experiments.

1.2 Automated Evaluation Pipeline

The submission will be evaluated based on the following weighted criteria:

1. **Forward Pass Verification (10 Marks):** The autograder will load fixed weights and verify that your model produces the correct output logits for a standard input tensor.
2. **Gradient Consistency (10 Marks):** Analytical gradients from your `backward()` pass will be compared against numerical gradients. A tolerance of 10^{-7} is required for full marks.
3. **Training Functionality (15 Marks):** The `train.py` script will be executed with varying hyperparameters to ensure robustness and correct logging (e.g., to Weights & Biases).
4. **Private Test Performance (10 Marks):** Your `best_model.npy` will be evaluated against a held-out private dataset to verify generalization.
5. **Code Quality (5 Marks):** Assessment of internal documentation (comments), `README.md` clarity, and adherence to standard coding styles.

2 Weights & Biases Report (50 Marks)

You must submit a public W&B report link. Your report must contain experimental evidence and written answers to the following questions:

2.1 Data Exploration and Class Distribution (3 Marks)

Log a W&B Table containing 5 sample images from each of the 10 classes in the dataset. Identify any classes that look visually similar in their raw form. How might this visual similarity impact your model?

2.2 Hyperparameter Sweep (6 Marks)

Perform a W&B Sweep with at least 100 runs, varying hyperparameters. Using the Parallel Coordinates plot, identify which hyperparameter had the most significant impact on validation accuracy. What was your best-performing configuration?

2.3 The Optimizer Showdown (5 Marks)

Compare the convergence rates of all 6 optimizers using the same architecture (3 hidden layers, 128 neurons each, ReLU activation). Which optimizer minimized the loss fastest in the first 5 epochs? Theoretically, why does Adam or Nadam often outperform standard SGD on image classification?

2.4 Vanishing Gradient Analysis (5 Marks)

Fix the optimizer to *Adam* and compare *Sigmoid* and *ReLU* for different network configurations. Log the gradient norms for the first hidden layer. Do you observe the vanishing gradient problem with Sigmoid? Provide a plot to support your observation.

2.5 The "Dead Neuron" Investigation (6 Marks)

Using **ReLU** activation and a high learning rate (e.g., 0.1), monitor the activations of your hidden layers. Find a run where the validation accuracy plateaus early. Look at the distribution of your activations. Can you identify "dead neurons" (neurons that output zero for all inputs)? Compare this run with a **Tanh** run and explain the difference in convergence based on the gradients you observed.

2.6 Loss Function Comparison (4 Marks)

Compare the training curves of five models: one using **Mean Squared Error (MSE)** and one using **Cross-Entropy**. Use the same architecture and learning rate for both. Which loss function converged faster? Theoretically, why is Cross-Entropy better suited for multi-class classification when paired with a Softmax output?

2.7 Global Performance Analysis (4 Marks)

Create an overlay plot showing the **Training vs. Test Accuracy** across every run you attempted during your hyperparameter search. Identify the runs that achieved high training accuracy but poor test accuracy. What does this gap indicate about the model?

2.8 Error Analysis (5 Marks)

Plot a **Confusion Matrix** for your best-performing model on the test set. For more marks: Beyond the standard matrix, provide a creative visualization of your model's failures.

2.9 Weight Initialization & Symmetry (7 Marks)

Compare two training runs with the following initialization strategies:

1. **Zeros Initialization:** All weights and biases are set to 0.
2. **Xavier Initialization:** Weights are sampled from a distribution with a specific variance.

In your W&B report, create a line plot showing the **gradients** of 5 different neurons within the *same* hidden layer for both runs over the first 50 training iterations.

1. In the “Zeros” run, you will notice that the gradients for all neurons within a layer are identical (the lines will overlap perfectly). Why does this “symmetry” prevent the network from learning complex, distinct features?
2. Even if the total loss decreases slightly, what is happening to the individual neurons? Use your gradient plots to explain why “Symmetry Breaking” (initializing with different values) is a mathematical necessity for a Multi-Layer Perceptron to function.

2.10 The Fashion-MNIST Transfer Challenge (5 Marks)

Note: All previous experiments were on digits (MNIST). Now, consider the Fashion-MNIST dataset.

- **The Scenario:** You have a limited computation budget. You are allowed only **3 hyperparameter configurations** to achieve the highest possible accuracy on Fashion-MNIST.
- **Task:** Based strictly on your learnings from the MNIST experiments, which 3 specific configurations (Architecture + Optimizer + Activation) would you choose? Report the accuracies obtained. Did the configuration that worked best for digits also work best for clothing? Justify why the complexity of the dataset affects your choice of hyperparameters.