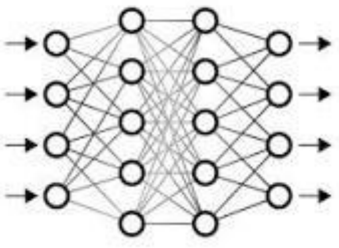
Fundamentals-First Neural Network: A From-Scratch AI System for Handwritten Digit Recognition



Abstract

As AI systems become increasingly abstracted through high-level libraries and autoML tools, many students and professionals lose touch with the core fundamentals of how machine learning models actually work. In this project, we tackle that gap by building a complete neural network **from scratch**, using only NumPy, to classify handwritten digits using the MNIST dataset. This endeavor required manually implementing every core aspect of a feedforward neural network, including data preprocessing, weight initialization, activation functions like ReLU and softmax, forward propagation, loss computation, backpropagation, and gradient descent. By forgoing high-level frameworks such as TensorFlow or PyTorch, this project provides an educational yet powerful look into how real AI models operate under the hood. The resulting system achieves competitive accuracy on the MNIST test set while maintaining full transparency, educational value, and technical control.

Introduction

To address this gap, this project focuses on building a **fully functional neural network model from the ground up** using only **NumPy**, trained and tested on the popular **MNIST handwritten digit classification dataset**. The MNIST dataset provides a classic benchmark problem — classifying 28×28 grayscale images into 10 digit classes (0–9).
This from-scratch approach helps students, researchers, and educators alike to truly grasp the inner mechanics of AI models, specifically how neurons interact, how gradients flow, and how errors are propagated and corrected — step-by-step

Proposed Solution:

The solution is a **single-agent, feedforward neural network system**, trained using supervised learning, implemented entirely in NumPy without any external ML libraries. The model was designed with the following components, all developed from scratch:

- **Data Preprocessing:** Normalizing and flattening MNIST images and splitting the data into training and testing sets using manual logic.
- **Architecture:** Multi-layer neural network (configurable layers) with customizable number of hidden units.

- **Activation Functions:** ReLU for hidden layers and softmax for output layer.
- **Loss Function:** Categorical Cross-Entropy implemented manually.
- **Training Process:** Forward propagation, manual gradient computation via backpropagation, and weight updates using mini-batch gradient descent.
- **Evaluation:** Manual accuracy and loss tracking on the test set after each epoch.

This implementation enforces a deep understanding of how networks process data, learn through iterative optimization, and generalize over unseen data.

PEAS for Neural Network Project

Component	Description
Performance	The agent will be assessed based on its ability to accurately predict outputs from the input data. This can be measured through metrics such as **accuracy** (currently 89%), **precision**, **recall**, and **F1-score** for a more holistic view of the model's performance. The model's generalization capability (how well it performs on unseen data) will also be considered.
Environment	The environment consists of the **dataset** (such as images, numerical data, or text), which the neural network learns from. It also includes the **training environment** (i.e., computing hardware, Python libraries like NumPy, Pandas, Matplotlib, and scikit-learn, and any external data sources).
Actuators	In this case, the actuator is the **neural network itself**, which "changes the environment" by adjusting its weights and biases through the backpropagation algorithm during training. The output of the network (the predictions) also represents a transformation of the environment.
Sensors	The **inputs to the neural network** act as sensors. These inputs are typically the data the model uses to make predictions (e.g., images, numerical features, etc.). The network senses the environment through these data points, which are processed by the model during both the training and inference phases.

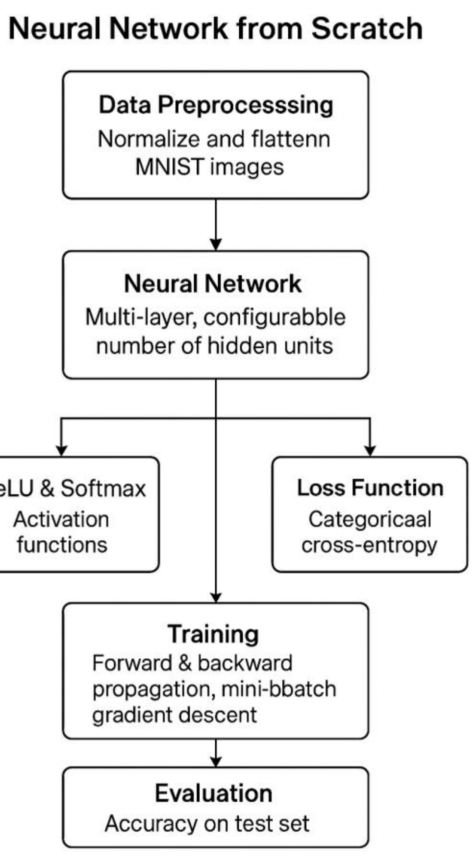
ODESDA Environment Properties

Property	**Type**	**Justification**
Observable/ Partially	**Fully Observable**	**All inputs (images) are completely available at every step.**
Deterministic/ Stochastic	**Deterministic**	**Same input and weights produce the same output; no stochasticity unless manually introduced.**
Episodic/ Sequential	**Episodic**	**Each input (image) is processed independently during training.**
Static/ Dynamic	**Static**	**Dataset doesn’t change; model trains on fixed data.**
Discrete/ Continuous	**Discrete**	**Output classes (digits 0–9) are discrete labels.**
Agent	**Single Agent**	**The model acts alone to perform classification without interaction or coordination with others.**

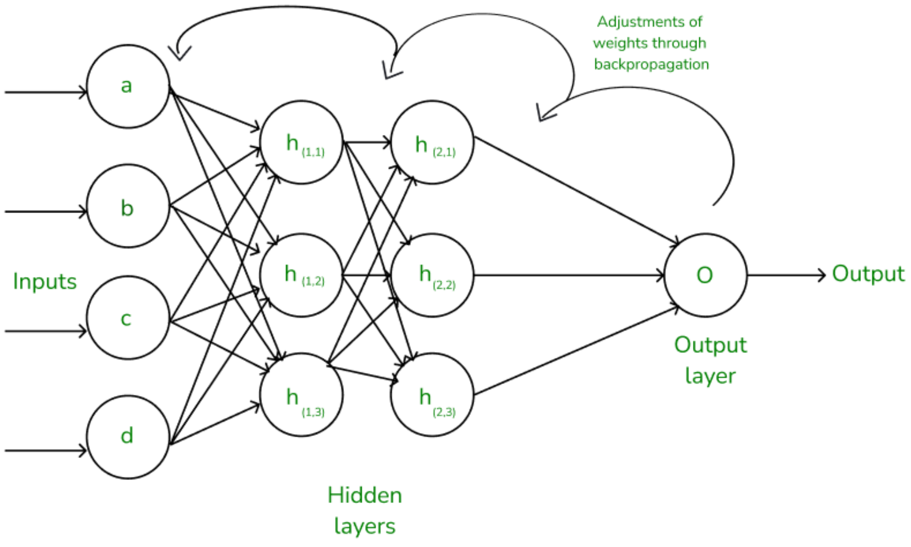
The model is considered as **Learning agent** because:

- **Learning element**: The neural network (model) itself.
- **Performance element**: The neural network's ability to make predictions.
- **Critic**: The loss function or evaluation metric.
- **Problem generator**: The data sampling or augmentation process that introduces new learning experiences during training.

Figure 1: NN Architecture


Neural Network from Scratch
Data Preprocesssing
Normalize and flattenn MNIST images
Neural Network
Multi-layer, configurabble number of hidden units
ReLU & Softmax
Activation functions
Loss Function
Categoricaal cross-entropy
Training
Forward & backward propagation, mini-bbatch gradient descent
Evaluation
Accuracy on test set

Neural Network Workflow:



Used Technologies[4]:

Technology	Usage
NumPy	Used for building the neural network from scratch, including matrix operations, forward and backward propagation, and gradient calculations.
Pandas	Used for reading and processing datasets efficiently.
Matplotlib	Used for plotting test images and visualizing predictions to assess the model's performance.
Scikit-learn	Used for accessing and loading datasets from various sources, simplifying data handling

Team Members

Member Name	Faculty Year
Wahb Mohamed Elsayad	Third year of Computer Science
Yahia Sherif Ibrahim	Third year of Computer Science
Yousef Ali Mansour	Third year of Computer Science

Future Work

While the current implementation of the neural network achieves an accuracy of 89%, there is significant potential for improvement. Several steps can be taken to enhance both the model's performance and its functionality:

1. **Model Optimization:**
   - **Hyperparameter Tuning:** Further tuning of hyperparameters such as learning rate, batch size, and number of layers could lead to improved model performance.
   - **Advanced Architectures:** Exploring more complex neural network architectures such as Convolutional Neural Networks (CNNs) could increase accuracy, especially if the task involves image data.
2. **Data Augmentation:**
   - If the dataset is limited, implementing data augmentation techniques could help by artificially expanding the training set and improving generalization.

3. **Regularization Techniques:**
    - Incorporating regularization methods like dropout or L2 regularization could prevent overfitting and help the model perform better on unseen data.
4. **Transfer Learning:**
    - Using pre-trained models for feature extraction and fine-tuning them for the specific task could significantly improve performance, especially in more complex tasks or with limited data.
5. **Evaluation Metrics:**
    - In addition to accuracy, exploring other evaluation metrics such as precision, recall, and F1-score could provide a more comprehensive understanding of the model's performance, especially in imbalanced datasets.
6. **Real-World Testing:**
    - Implementing the model in real-world scenarios with live data could reveal additional challenges that may not have been captured during testing. This will help in adapting the model for practical applications.
7. **Deployment and Scalability:**
    - Investigating ways to deploy the model efficiently, such as using cloud platforms or optimizing for real-time predictions, could make the model more accessible and applicable in production environments.