End-to-End Implementation of a Fully Connected Neural Network for Fashion-MNIST Classification Without Deep Learning Frameworks

M.Srinija

Hyderbad, India

3 August, 2025

I. Abstract

This project presents an end-to-end implementation of a fully connected neural network for Fashion-MNIST classification using only NumPy, without relying on any deep learning frameworks like TensorFlow or PyTorch. By building each component from scratch—forward propagation, backpropagation, dropout, regularization, and optimizers—it offers a deep dive into how neural networks function under the hood. Fashion-MNIST, with its diverse set of 28×28 grayscale images across 10 clothing categories, serves as an ideal benchmark: more challenging than MNIST yet lightweight and accessible. The model achieves high accuracy, reinforcing both practical understanding and foundational knowledge of deep learning mechanics.

Keywords: Image Classification, Fashion-MNIST, Neural Network from Scratch, NumPy

II. Dataset: Fashion-MNIST

The Fashion-MNIST dataset, developed by Zalando Research, is a more challenging alternative to the classic MNIST dataset. It contains 70,000 grayscale images of size 28×28 pixels, categorized into 10 classes of everyday clothing items such as T-shirts, dresses, sneakers, and bags. Designed to better represent real-world classification tasks, it retains the simplicity of MNIST while introducing greater visual complexity. For this project, each image was flattened into a 784-dimensional vector and normalized to a [0, 1] scale. Labels were one-hot encoded, and the dataset was shuffled and batched to enable efficient mini-batch gradient descent during training.

III. Method

A. Model Architecture

The model consists of an input layer with 784 neurons (for flattened 28×28 images), two hidden layers with 196 and 64 neurons respectively using ReLU activation, and a 10-neuron output layer with softmax activation for multi-class classification. This fully connected architecture enables hierarchical feature learning, where the first hidden layer captures abstract patterns and the second refines them into higher-level features. The output layer translates these features into class probabilities, allowing the model to make predictions across the 10 fashion categories.

B. Forward Propagation

Forward propagation computes the output by passing input data through each layer in sequence. Linear transformations ($Z = W \cdot A + b$) are followed by non-linear activations (ReLU in hidden layers, softmax at the output). Dropout is optionally applied during training to randomly deactivate a portion of neurons, preventing overfitting. Each layer's computations are stored as caches, which are later used during backpropagation. This process produces class probabilities that reflect the model's confidence in its predictions for each input sample.

C. Loss Function

Categorical cross-entropy loss is computed between predicted softmax probabilities and true one-hot encoded labels. A small epsilon is added for numerical stability. L2 regularization is incorporated to penalize large weights and reduce overfitting.

D. Backpropagation

Backpropagation computes gradients of the loss with respect to weights and biases using the chain rule. Starting from the output layer, gradients are calculated layer by layer in reverse order, using stored caches. For ReLU, the derivative zeroes out negative inputs; for softmax combined with cross-entropy, a simplified derivative is used. Dropout masks are reapplied during gradient computation. L2 regularization is added to weight gradients to penalize large weights, promoting generalization. Gradient clipping is also used to prevent instability from exploding gradients.

E. Optimization

The model supports multiple optimization strategies: Vanilla SGD, SGD with Momentum, and Adam. In Vanilla SGD, weights are updated using basic gradient descent. Momentum adds a velocity term to accelerate convergence on slopes and reduce oscillations. Adam combines momentum with adaptive learning rates, adjusting updates based on both past gradients and their squared values. These methods are modular and selectable. Among them, Adam yielded the best performance in terms of speed and stability, helping the model reach high accuracy within fewer epochs.

F. Regularization & Dropout

To prevent overfitting, L2 regularization and dropout were integrated into the model. L2 regularization penalizes large weight values by adding a scaled sum of squared weights to the loss function, encouraging simpler models. Dropout randomly deactivates neurons during training, forcing the network to develop redundant, robust representations. A keep probability of 0.9 was used, balancing stability and regularization. These techniques reduced training-test accuracy gaps, stabilized convergence, and improved generalization on unseen Fashion-MNIST data, especially during longer training sessions.

IV. Features Implemented from Scratch

A. Weight & Bias Initialization: Initialized using He initialization for ReLU stability and prevents vanishing/exploding gradients at deep layers.

$$\sigma = \sqrt{2/fan_{in}}$$

B. Forward Propagation: Computes neuron activations layer-by-layer and stores intermediate results (Z, A) in caches for backpropagation

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$

$$A^{l} = activation(Z^{l})$$

C. Activation Functions: ReLU, Softmax, Tanh add non-linearity and output probability computation and crucial for deep learning expressiveness and classification.

$$\text{ReLU: } A = \max(0, Z)$$

$$\text{Softmax: } A_i = \frac{e^{Z_i}}{\sum_j e^{Z_i}}$$

D. Backpropagation: Implemented gradient computation using chain rule.

$$\frac{\partial L}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l]}$$

E. Dropout: For forward, randomly drops neurons during training and for backward, it applies same masks to gradients

F. L2 Regularization: Adds penalty term to the loss function and controls overfitting by shrinking large weights

$$L = CE_{loss} + \frac{\lambda}{2m} \sum_{l=1}^{L} \left|\left| W^{[l]} \right|\right|^2$$

G. Loss Function: Cross-Entropy for Softmax measures prediction error between ground truth and softmax output and provides smooth gradients for optimization

$$L = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij} + \varepsilon$$

H. Mini-Batch Creation: Randomly shuffled and batched training data that makes training faster and generalization better

I. Early Stopping: Monitors performance and halts when improvement stops which saves resources and prevents overfitting

J. Optimizers:

- SGD

$$\theta = \theta - \alpha . \nabla_\theta J$$

- Momentum

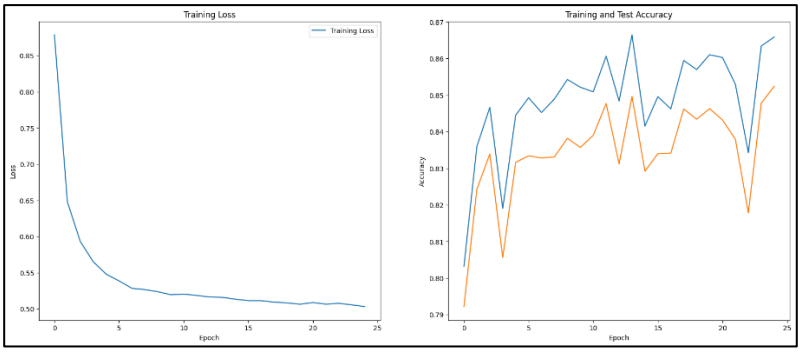
$$v = \beta v + (1 - \beta) \nabla_\theta J, \ \theta = \theta - \alpha v$$

- Adam

$$Uses: v, s, \quad \theta = \theta - \alpha . \frac{\hat{v}}{\sqrt{\hat{s}} + \varepsilon}$$

V. Results & Discussion

The neural network achieved a peak test accuracy of ~85% on the Fashion-MNIST dataset using the Adam optimizer with dropout and L2 regularization. Dropout and weight decay significantly improved generalization, while mini-batching stabilized convergence. Training loss decreased steadily across epochs, and early stopping prevented overfitting. Experiments showed that Adam converged faster than SGD variants. These results demonstrate that even without deep learning libraries, a well-constructed architecture and thoughtful optimization techniques can yield strong performance on real-world classification tasks.



VI. Conclusion

This project demonstrates the feasibility and effectiveness of building a fully functional deep learning model from scratch using only NumPy. By implementing each component—forward propagation, backpropagation, dropout, regularization, and optimizers—we gained a deep, practical understanding of neural network internals. Applied to the Fashion-MNIST dataset, the model achieved competitive accuracy while maintaining flexibility and interpretability. Through this process, we not only learned how deep learning works at a fundamental level but also proved that strong performance can be achieved without relying on prebuilt frameworks, making it an excellent learning experience in both theory and practice.

VII. References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," J. Mach. Learn., pp. 1929–1958, 2014.

[2] M. A. Nielsen, Neural Networks and Deep Learning, 2015. Available: http://neuralnetworksanddeeplearning.com

[3] C. R. Harris et al., "Array programming with NumPy," Nature, 2020. Available: https://numpy.org/