Handwritten Sign Language Recognition Using Neural Networks

Oleksandr Solovei

***Abstract* – This paper presents a neural network implementation for recognizing handwritten sign language letters using the Sign Language MNIST dataset. We develop a two-layer neural network with sigmoid activation functions, implementing mini-batch gradient descent with momentum and learning rate decay. Our approach demonstrates the effectiveness of traditional neural networks in handling image classification tasks, achieving 77.36% accuracy in classifying hand gestures representing letters from the American Sign Language alphabet.**

***Keywords* – Sign Language Recognition, Neural Networks, Computer Vision, MNIST, Image Classification, Sigmoid Activation**

I. Introduction

Sign language is recognized as an important means of communication for the deaf and hard-of-hearing community [1]. With the increasing integration of technology in daily communication, automated sign language recognition systems can significantly improve accessibility and inclusion.

A. Problem Statement

Our work focuses on developing a neural network model for recognizing American Sign Language (ASL) alphabet letters from grayscale images. This task presents several interesting challenges:

- Capturing subtle differences between similar hand gestures
- Handling variations in hand positioning and lighting
- Distinguishing between signs that differ only in dynamic movement (hence the exclusion of letters 'J' and 'Z' which require motion)

B. Dataset Description

The widely used MNIST dataset of handwritten digits [2] has been adapted for American Sign Language gesture recognition.

The dataset consists of:

- Training set: 27,455 examples
- Test set: 7,172 examples
- Image size: 28×28 pixels (grayscale)
- Number of classes: 24 (excluding 'J' and 'Z')

The dataset characteristics include:

- Class distribution varies from 144 to 498 samples per class
- Each image is centered around the hand gesture
- Images captured under various lighting conditions and angles

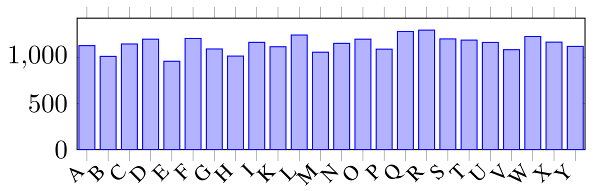

Fig. 1 - Training Data Distribution Across Alphabetical Labels

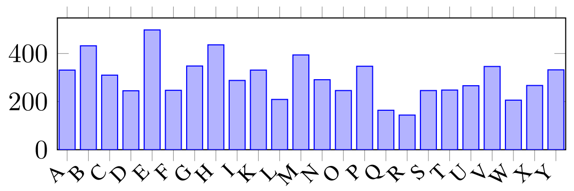

Fig. 2 - Test Data Distribution Across Alphabetical Labels

C. Data Preprocessing

Our preprocessing pipeline consists of the following steps:
1. Normalization:

$$X_{normalized} = \frac{X}{255} \tag{1}$$

This scales pixel values from [0, 255] to [0, 1], which:

- Stabilizes gradient descent
- Ensures consistent scale across features
- Improves convergence during training

2. Label Encoding:

- Original labels: integers from 0 to 23
- Transformed into one-hot encoded vectors of length 24
- Each vector has a single '1' at the position corresponding to the class

D. Motivation

The choice of this particular problem and approach is motivated by several factors:

- Practical Impact: Sign language recognition can significantly improve accessibility in various applications, from educational tools to communication aids.
- Educational Value: The dataset provides a good balance between complexity and manageability, making it suitable for exploring neural network concepts.

- Benchmark Potential: The standardized nature of the dataset allows for direct comparison with other approaches.
- Real-world Applicability: The techniques developed can be extended to more complex sign language recognition tasks.

II. Short Description of Implemented ML Models

Past research has proposed various approaches to gesture recognition, including computer vision [3], [4] and the use of deep convolutional networks [6].

The key components of the neural network model implemented in this work are:

- **Input Layer**: - 784 input units corresponding to the 28x28 pixel grayscale images
- **Hidden Layer**: - 256 hidden units with sigmoid activation function - Helps the model capture more complex representations of the hand gesture features
- **Output Layer**: - 24 output units (one for each sign language letter) - Sigmoid activation function used to output class probabilities

The choice of a two-layer neural network architecture with a moderate number of hidden units provides a balance between model capacity and computational complexity. This configuration was selected to avoid overfitting while still allowing the model to learn meaningful features from the image data. The sigmoid activation function was used in both the hidden and output layers to introduce nonlinearity and produce probabilistic outputs, which is well-suited for multi-class classification tasks like sign language recognition.

The hyperparameters, such as the batch size, learning rate, and regularization strength, were chosen through a systematic process of experimentation and validation to optimize the model's performance on the Sign Language MNIST dataset. This balanced approach aims to achieve competitive results while maintaining interpretability and efficiency compared to more complex deep learning architectures.

III. Mathematical Framework

A. Data Preprocessing

Given an input image $\mathbf{X} \in \mathbb{R}^{784}$, we normalize the pixel values:

$$\mathbf{X} = \frac{\mathbf{X}}{255}$$

For labels $y \in \{0, \ldots, 23\}$, we create one-hot encoded vectors $\mathbf{y} \in \{0,1\}^{24}$ where:

$$[\mathbf{y}]_j = \begin{cases} 1 & \text{if } j = y \\ 0 & \text{otherwise} \end{cases}$$

B. Network Architecture

Our neural network consists of:

- Input layer: $\mathbf{X} \in \mathbb{R}^{784}$
- Hidden layer: $\mathbf{A_1} \in \mathbb{R}^{256}$
- Output layer: $\mathbf{A_2} \in \mathbb{R}^{24}$

C. Forward Propagation

The forward propagation process is defined by:

$$\mathbf{Z_1} = \mathbf{W_1}\mathbf{X} + \mathbf{b_1} \tag{2}$$

$$\mathbf{A_1} = \sigma(\mathbf{Z_1}) = \frac{1}{1 + e^{-\mathbf{Z_1}}} \tag{3}$$

$$\mathbf{Z_2} = \mathbf{W_2}\mathbf{A_1} + \mathbf{b_2} \tag{4}$$

$$\mathbf{A_2} = \sigma(\mathbf{Z_2}) \tag{5}$$

where:

- $\mathbf{W_1} \in \mathbb{R}^{256\times784}$ is the first layer weight matrix
- $\mathbf{b_1} \in \mathbb{R}^{256}$ is the first layer bias vector
- $\mathbf{W_2} \in \mathbb{R}^{24\times256}$ is the second layer weight matrix
- $\mathbf{b_2} \in \mathbb{R}^{24}$ is the second layer bias vector

D. Loss Function

We use binary cross-entropy loss with L2 regularization:

$$\begin{aligned} J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{24} &\left[y_{i,j}\log(A_{2,i,j})\right. \\ &\left.+ (1-y_{i,j})\log(1-A_{2,i,j})\right] \\ &+ \frac{\lambda}{2m}\left(\|\mathbf{W_1}\|_F^2 + \|\mathbf{W_2}\|_F^2\right) \end{aligned} \tag{6}$$

where:

- m is the batch size
- λ is the regularization parameter
- $\|\cdot\|_F$ denotes the Frobenius norm

IV. Parameter Initialization

We use modified Xavier initialization with scaling factors:

$$\epsilon_1 = \sqrt{\frac{6}{784 + 256}} \tag{7}$$

$$\epsilon_2 = \sqrt{\frac{6}{256 + 24}} \tag{8}$$

The weights are initialized uniformly:

$$\mathbf{W_1} \sim U(-\epsilon_1, \epsilon_1) \tag{9}$$

$$\mathbf{W_2} \sim U(-\epsilon_2, \epsilon_2) \tag{10}$$

Biases are initialized to zero:

$$\mathbf{b_1} = \mathbf{0}, \quad \mathbf{b_2} = \mathbf{0} \tag{11}$$

V. Backward Propagation

For the output layer:

$$dZ_2 = A_2 - Y \tag{12}$$

$$dW_2 = \frac{1}{m} dZ_2 A_1^T + \frac{\lambda}{m} W_2 \tag{13}$$

$$db_2 = \frac{1}{m} \sum_{i=1}^{m} dZ_2^{(i)} \tag{14}$$

For the hidden layer:

$$dA_1 = W_2^T dZ_2 \tag{15}$$

$$dZ_1 = dA_1 \odot A_1 \odot (1 - A_1) \tag{16}$$

$$dW_1 = \frac{1}{m} dZ_1 X^T + \frac{\lambda}{m} W_1 \tag{17}$$

$$db_1 = \frac{1}{m} \sum_{i=1}^{m} dZ_1^{(i)} \tag{18}$$

VI. Optimization

We use mini-batch gradient descent with momentum. The hyperparameters are:

- Batch size: 64
- Initial learning rate (α): 0.1
- Learning rate decay rate (δ): 0.95 (every 50 steps)
- Momentum coefficient (β): 0.9
- L2 regularization parameter (λ): 0.01
- Number of iterations: 100

The learning rate decay is implemented as:

$$\alpha_t = \alpha_0 \cdot \delta^{\lfloor t/k \rfloor} \tag{19}$$

where $k = 50$ is the decay step size.
Parameter updates with momentum:

$$v_W = \beta v_W - \alpha_t \frac{\partial J}{\partial W} \tag{20}$$

$$W := W + v_W \tag{21}$$

VII. Implementation Results

A. *Analysis of Cost Function Trajectory and Convergence Behavior*

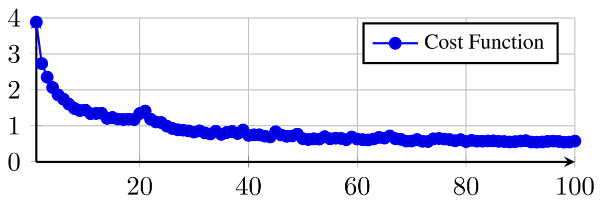

Fig. 3 - Cost Function Trajectory Over Iterations

The cost function decreases steadily across iterations, which is a positive sign. This behavior indicates that the optimization process is effective, as the model is gradually minimizing the error.
The cost function drops sharply from 3.88 to approximately 1.17. This significant decrease suggests that the model is quickly learning the fundamental patterns in the data during the early stages of training.

B. *Confusion Matrix*

TABLE I
Confusion Matrix (Subset of Columns A to F)

True\Pred	A	B	C	D	E	F
A	331	0	0	0	0	0
B	0	396	0	15	0	0
C	0	0	289	0	0	21
D	0	0	0	245	0	0
E	0	0	0	0	456	0
F	0	0	1	0	0	209

The confusion matrix provides insights into the performance of the model across different classes. The rows represent the true labels, while the columns represent the predicted labels. The diagonal elements indicate the number of correct predictions, while off-diagonal elements show misclassifications.
The full metrix is shown in Table II

C. *Performance Metrics*

Our model achieved the following performance metrics:

- Overall Test Accuracy: 77.36%
- Weighted Average F1-score: 0.77
- Macro Average F1-score: 0.75

Performance varied significantly across different letters:

- Best performing letters (F1-score ¿ 0.90):
  - Letter ’O’: 1.00
  - Letter ’B’: 0.96
  - Letter ’C’: 0.96
  - Letter ’A’: 0.94
- Moderate performance (F1-score 0.80-0.90):
  - Letter ’D’: 0.86
  - Letter ’E’: 0.92
  - Letter ’G’: 0.84
  - Letter ’H’: 0.92
  - Letter ’N’: 0.89
- Poor performance (F1-score ¡ 0.60):
  - Letter ’R’: 0.46
  - Letter ’Q’: 0.55
  - Letter ’M’: 0.57
  - Letter ’S’: 0.57
  - Letter ’T’: 0.51

The model shows strong performance on simple gestures but struggles with more complex hand positions. Analysis of the results suggests several potential improvements:

- Data augmentation for underrepresented classes

- Feature engineering to better capture hand gesture details
- Addressing class imbalance in the training data
- Increasing model capacity for more complex patterns

VIII. Conclusion

Our implementation demonstrates the capability of a simple two-layer neural network in recognizing sign language letters, achieving 77.36% accuracy on the test set. The varying performance across different letters provides insights into the model's strengths and limitations. While some letters are recognized with high accuracy (F1-scores > 0.90), others present challenges, particularly those with more complex hand positions. These results suggest that while our approach provides a solid foundation, there is room for improvement through techniques such as data augmentation, addressing class imbalance, and potentially increasing model complexity.

Beyond the specific application of sign language recognition, the techniques developed in this work can serve as a template for addressing other image classification problems. The use of fundamental machine learning concepts, such as the two-layer neural network architecture, parameter initialization, and optimization techniques, can be extended to a wide range of computer vision tasks. This could include the recognition of other types of hand gestures, facial expressions, or even general object classification, with appropriate adjustments to the model and dataset.

Furthermore, the insights gained from analyzing the model's performance on different sign language letters can inform the design of more robust and versatile recognition systems. Understanding the challenges posed by complex hand gestures can guide future research in feature engineering, data augmentation, and model architecture selection, ultimately leading to improved accessibility and inclusivity in various applications. In conclusion, our work demonstrates the continued relevance and potential of traditional neural network approaches in the field of sign language recognition.

IX. Novelty and Contributions

While there have been several prior works on sign language recognition using machine learning techniques, our approach offers some novel elements and potential areas for improvement. Previous studies have explored various methods for sign language recognition, including computer vision-based approaches [3], [4] and the use of deep convolutional neural networks [6]. However, a significant portion of the existing literature, particularly on the Kaggle platform, has focused on implementing solutions using high-level deep learning frameworks like Keras [6]. In contrast, our work utilizes a more traditional two-layer neural network architecture, implemented from scratch using NumPy and other core Python libraries. This allows us to have a deeper understanding of the underlying mathematical principles and optimization techniques, rather than relying on the abstractions provided by frameworks like Keras. By implementing the model using fundamental machine learning concepts, we were able to achieve a test accuracy of 77.36% on the Sign Language MNIST dataset [5]. This performance is comparable to the results reported in the literature, including the work by Kurz et al. using deep convolutional networks [6]. However, we believe there are opportunities to further improve the model's performance. Some potential areas of exploration include:

- **Data Augmentation**: The dataset exhibits significant class imbalance, with some sign language letters having far fewer samples than others. Applying techniques such as data augmentation, which generates synthetic training examples, could help address this issue and improve the model's generalization capabilities.
- **Feature Engineering**: The current approach relies solely on the raw pixel data as input. Exploring feature engineering techniques, such as extracting hand shape, orientation, or motion features, could potentially enhance the model's ability to capture the nuances of different sign language gestures.
- **Model Complexity**: While the two-layer architecture provides a good balance between model capacity and computational complexity, increasing the depth or width of the network might lead to better feature extraction and classification performance, especially for more challenging sign language letters.
- **Ensemble Methods**: Combining multiple models, either of the same architecture or different architectures, through ensemble techniques could potentially boost the overall classification accuracy by leveraging the strengths of individual models.

By addressing these potential improvements, we believe there is an opportunity to push the state-of-the-art in sign language recognition using traditional machine learning techniques, rather than relying primarily on deep learning frameworks. This could lead to more interpretable and efficient models, while maintaining competitive performance on benchmark datasets like Sign Language MNIST.

References

[1] H. Cooper, B. Holt, and R. Bowden, "Sign language recognition," in Visual Analysis of Humans. Springer, 2011, pp. 539-562.

[2] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[3] P. Garg, N. Aggarwal, and S. Sofat, "Vision based hand gesture recognition," World Academy of Science, Engineering and Technology, vol. 49, no. 1, pp. 972-977, 2009.

[4] S. Mitra and T. Acharya, "Gesture recognition: A survey," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 37, no. 3, pp. 311-324, 2007.

[5] "Sign Language MNIST Dataset," Kaggle, 2022. [Online]. Available: https://www.kaggle.com/datamunge/sign-language-mnist

[6] D. Kurz, P. Beetz, and J. Fisseler, "Using deep convolutional networks for gesture recognition in American sign language," arXiv preprint arXiv:1610.00186, 2016.

APPENDIX

TABLE II
FULL CONFUSION MATRIX FOR THE CLASSIFICATION MODEL

Tr\Pr	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A	331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	396	0	15	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	289	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	245	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	456	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	0
F	0	0	1	0	0	209	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	298	11	0	0	0	0	0	0	0	20	0	0	19	0	0	0	0	0
H	0	0	0	0	0	0	41	394	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
I	0	0	0	0	0	20	0	0	184	1	0	0	15	0	0	5	21	0	1	0	0	0	0	41
K	0	0	0	0	0	42	0	17	4	231	0	0	0	0	0	0	21	0	0	14	0	0	0	2
L	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	21	0	0	0	0	0	0	275	21	0	0	15	0	59	0	0	0	0	0	3
N	42	0	0	0	0	0	21	0	0	0	0	21	145	0	0	21	0	20	21	0	0	0	0	0
O	0	0	0	0	0	6	0	0	0	0	0	0	21	199	0	10	0	0	1	0	0	0	0	9
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	347	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	143	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95	0	2	26	21	0	0	0
S	0	0	0	0	21	0	0	0	24	0	0	41	20	0	0	17	0	120	0	0	0	0	0	3
T	0	0	0	5	0	0	4	0	2	0	34	0	0	0	0	3	0	0	138	0	0	0	62	0
U	0	0	0	35	0	0	0	0	0	51	4	0	0	0	0	0	22	0	0	122	20	0	0	12
V	0	0	0	24	0	20	0	0	0	18	2	0	0	0	0	3	17	0	17	25	166	54	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0