American Sign Language Recognition Using Machine Learning Models

1st Solomonidi Eleftheria
Department of Electronics, Telecommunications and Informatics
University of Aveiro
Aveiro, Portugal
elefth@ua.pt, 130364

2nd Bilal el Hanane
Department of Mechanical Engineering
University of Aveiro
Aveiro, Portugal
bilal.elhanane@ua.pt, 130152

***Abstract*—Sign language is an important communication method for people who are deaf or unable to speak.The aim of this project is to classify images of American Sign Language (ASL) hand gestures using machine learning. The Sign Language MNIST dataset from Kaggle is used to classify 28×28 grayscale images of ASL hand signs and apply basic preprocessing while train different models focusing on neural networks. Firstly a baseline model is tested using Logistic Regression, and then is developed a neural network (MLP) with one hidden layer. Several hyperparameters such as hidden layer size, learning rate, and regularization strength are tuned to improve performance. Training and validation results are analyzed through accuracy metrics such as F1-scores, and loss curves. The final optimized MLP achieves near perfect accuracy on the training and validation sets, while the test performance shows the challenges of generalization. This projects shows how model complexity and hyperparameter selection can affect the performance of image-based gesture classification.**

***Index Terms*—machine learning, sign language recognition, neural networks, logistic regression, image classification, American Sign Language**

I. Introduction

American Sign Language (ASL) is an essential way for people who are deaf or cannot speak to communicate. Being able to automatically recognize hand signs can make technology more accessible, and improve interaction between humans and computers. This problem is meaningful and practical, and it offers a good example of how machine learning can be applied to help real users in everyday situations. To understand the complexity of the problem, we first train a Logistic Regression model as a simple baseline. We then develop a feedforward neural network (MLP) using a hidden layer with ReLU activation and the Adam optimizer. Several hyperparameters such as hidden layer size, learning rate, number of iterations, and regularization are systematically tuned to study how they affect performance. The goal of this work is to compare the two models, analyze their training behavior using loss curves, and evaluate generalization through validation and test accuracy.

A. Data Description, ML Problem, and Preprocessing

The project uses the Sign Language MNIST dataset from Kaggle, which contains grayscale images of American Sign Language (ASL) hand gestures as shown in "Fig. 1". Each image is 28×28 pixels, flattened into a vector of 784 input features. The goal is to classify each image into one of 24 output classes, corresponding to the letters A-Y (excluding J and Z, which require motion). Therefore, this is a multiclass classification problem where:

Fig. 1. Example of sample images.

- **Input:** a vector of 784 normalized pixel values
- **Output:** one class label from 0 to 23 (representing 24 hand gestures)

Preprocessing: Before training the models, the pixel values are normalized by dividing each value by 255.0 so that all features lie in the range $[0, 1]$. Normalization helps neural networks train more smoothly and prevents large gradients.

After normalization, the data is split again to create a separate validation set (10% of the training data). The split is *stratified*, ensuring that all classes remain equally represented in the training and validation subsets. The test dataset provided by Kaggle is kept separate for the final evaluation.

B. Machine Learning Models

1) Baseline Model: ***Logistic Regression****:* Logistic Regression was used as the first baseline model because it is quite simple to implement. Logistic Regression is a linear classification model. It takes the input features (the pixel values of the images) and tries to learn a set of weights that can separate the classes. For multi-class problems, it uses a "one-vs-rest" strategy. Each class gets its own linear decision boundary.

The model is simple and easy to train. Its main limitation is that it can only learn linear relationships, so its performance depends strongly on how well the classes can be separated using straight lines in the feature space.

As expected, logistic regression achieved reasonable accuracy but could not match the performance of neural networks, which are able to learn richer image features.

*2) **Neural Network Architecture (MLP):*** The MLP is a neural network model with one or more hidden layers. Unlike Logistic Regression, it can learn non-linear patterns because each hidden layer applies a non-linear activation function.

The model used in this project takes the flattened pixel values as input and passes them through one hidden layer with ReLU activation. The output layer uses softmax to produce probabilities for the 24 classes. The Adam optimizer is used for training, which adapts the learning rate during optimization and converges faster than basic gradient descent.

Additional information on common MLP architectures and hyperparameters can be found in [1]. In this project, only a subset of those hyperparameters was tuned, based on the needs of the dataset.

The main hyperparameters considered were the *size of the hidden layer*, the *learning rate*, the strength of *regularization* λ and the *maximum number of training iterations* (`max_iter`). These parameters strongly affect how the model learns. Because an MLP can learn more complex patterns than Logistic Regression, it can also overfit more easily, especially when trained for many iterations or when regularization is too small.

The diagram below, "Fig. 2", shows the structure of the network:

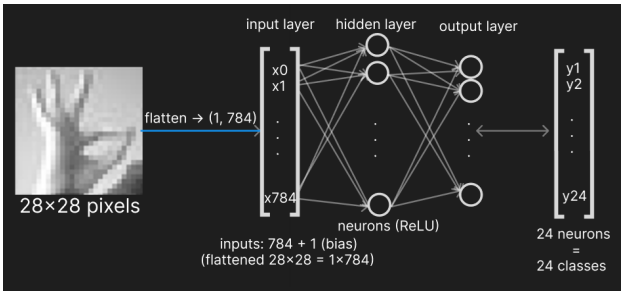

Fig. 2. Architecture of the MLP model, created in Figma.

Input Layer: Each input image is originally of size 28×28 pixels and is reshaped into a one-dimensional vector of length 784. A bias term is appended, forming the input vector:

$$x = (1, x_0, x_1, \ldots, x_{783}),$$

where the first component is the bias unit. This vector is fed into the first layer of the network.

Hidden Layer: The model uses a hidden layer with 64 units. This layer allows the network to learn patterns that logistic regression cannot, such as edges, curves, or pixel combinations that form meaningful shapes. After the hidden layer, the network produces a vector of activations, where each activation is computed as

$$h = \mathrm{ReLU}(W_1 x + b_1),$$

with W_1 and b_1 representing the weights and biases of the hidden layer. These activations capture non-linear patterns in the input image.

ReLU Activation: Each hidden neuron applies the ReLU (Rectified Linear Unit) activation function, "Fig. 3":

$$\mathrm{ReLU}(x) = \max(0, x),$$

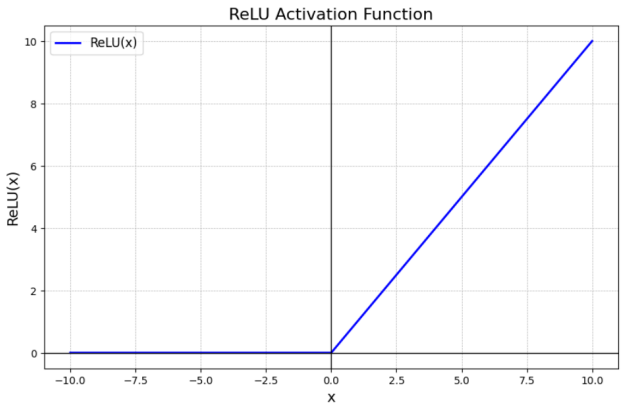

Fig. 3. ReLU activation function

Output Layer: The final layer contains 24 output neurons, one for each ASL class (meaning 1 class represents 1 letter). These outputs represent the class scores. After the output layer, the model computes the raw class scores:

$$z_2 = W_2 h + b_2,$$

and applies the Softmax function to obtain class probabilities.

Softmax converts the raw output scores into probabilities that sum to 1:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C=24} e^{z_j}}.$$

The predicted class is the one with the highest probability.

a) Training Objective (Cross-Entropy + Regularization): The model is trained using the cross-entropy loss for multiclass classification:

$$L_{\mathrm{CE}} = -\sum_{i=1}^{24} y_i \log(\hat{y}_i),$$

where y_i is the one-hot encoded true label. L2 regularization is added to prevent overfitting:

$$L_{\mathrm{reg}} = \lambda \left(\|W_1\|_2^2 + \|W_2\|_2^2 \right),$$

where λ is the regularization strength. The final training objective is:

$$L = L_{\mathrm{CE}} + L_{\mathrm{reg}}.$$

b) Optimization with Adam: The weights are optimized using the Adam algorithm (Adaptive Moment Estimation) [2], a variant of stochastic gradient descent. Adam adaptively adjusts the learning rate for each parameter using estimates of the first and second moments of the gradients. It maintains two exponential moving averages during training: the first moment (mean of gradients) and the second moment (uncentered

variance). These are used to adapt the learning rate for each parameter, resulting in more stable and efficient optimization.

The first moment estimate m_t accumulates past gradients similarly to classical momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$

where g_t is the gradient at step t. The second moment estimate v_t tracks the magnitude of the gradients:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2.$$

Here, m_t represents a smoothed direction of descent, while v_t adaptively scales the learning rate based on gradient variance. This enables Adam to accelerate when gradients consistently point in the same direction, avoid small fluctuations in the loss landscape, and converge more smoothly. By using these stabilized estimates rather than raw gradients, Adam yields more reliable and efficient training, especially for noisy or high-dimensional datasets.

C. Hyperparameter Tuning Approach

Each hyperparameter was tuned using a separate script, allowing the effect of individual parameters to be evaluated independently. After determining the optimal values for each hyperparameter, a final training script was executed using the combined optimized configuration.

1) Hidden Layer Size: The hidden-layer size directly affects convergence behaviour, generalization performance, and training cost. To select an appropriate size, models were trained using seven configurations:

$$h \in \{8,\ 16,\ 32,\ 64,\ 128,\ 256,\ 512\}.$$

All experiments used identical training settings (L2 regularization $\alpha = 0.0001$, early stopping, validation fraction 0.1, and a maximum of 200 iterations) to ensure comparability.

Summary of Observations:

- **Small sizes (8–16):** low training accuracy, low validation accuracy and F1, loss curves that stop improving early, overall underfitting.
- **Satisfied size (32):** overall good training and validation accuracy but takes more time than higher sizes.
- **Medium sizes (64–128):** best overall behaviour, including stable convergence, high validation accuracy, and reasonable training time.
- **Large sizes (256–512):** rapid overfitting, unstable loss curves, and substantially higher computational cost without improving generalization.

Conclusion: Hidden-layer sizes of **64 and 128** provided the best balance between model capacity, generalization, convergence stability, and training efficiency. These sizes were therefore selected for subsequent learning-rate tuning.

Results: Table I summarizes the performance metrics for all tested configurations.

TABLE I
PERFORMANCE OF DIFFERENT HIDDEN-LAYER SIZES.

Hidden	Iter	Time (s)	Train Acc	Val Acc	Val F1
8	18	3.60	0.04691	0.04916	0.00461
16	106	19.50	0.31402	0.30991	0.26759
32	200	75.16	0.91570	0.89913	0.89828
64	78	25.35	0.99988	0.99964	0.99964
128	75	35.49	0.99996	1.00000	1.00000
256	62	88.93	0.99996	0.99964	0.99964
512	43	149.10	0.99996	1.00000	1.00000

Loss Curve Visualization: Figure 4 provides a visual comparison of the training loss trajectories for all hidden-layer sizes. The curves illustrate the capacity-related behaviour described above: underfitting for very small networks, smooth and stable convergence for medium sizes (64–128), and rapid overfitting or unstable behaviour for larger networks.

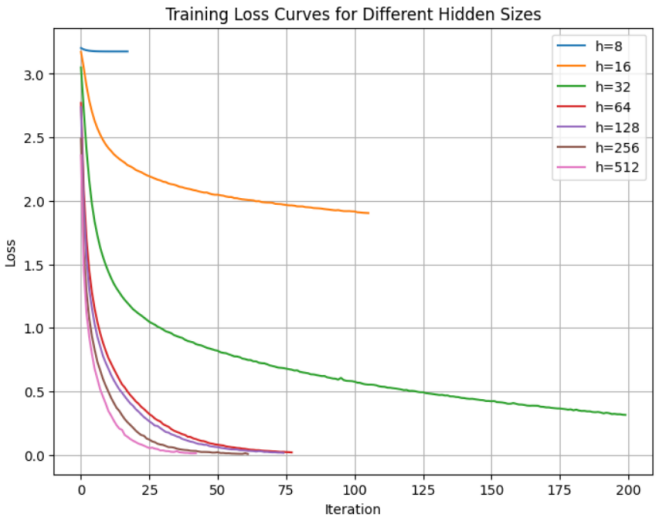

Fig. 4. Training loss curves for different hidden-layer sizes.

2) Learning Rate: The learning rate controls the step size of the optimizer (Adam). Illustrations adapted from [3],"Fig. 5", show how different learning rate values affect the gradient descent method during training. The parameter update rule for the Adam optimiser is then given by:

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

where η is the learning rate and ϵ is a small constant for numerical stability. These behaviors emphasize the importance of selecting an appropriate learning rate when training and tuning neural network models.

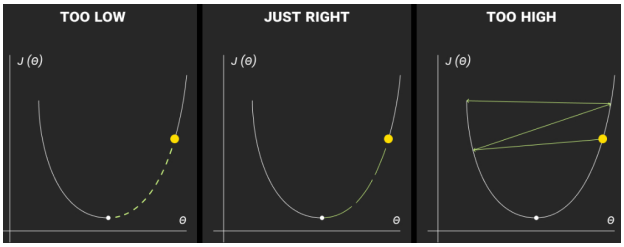

Fig. 5. Different Learning Rates, [3]

The loss–learning rate relationship can be interpreted through three characteristic regions:

- **Learning rate too small:** the loss decreases very slowly, indicating that the optimizer is taking steps that are too small to make meaningful progress. Although convergence is stable, the model may require many iterations to reduce the loss, resulting in unnecessarily long training times.
- **Learning rate in a good range:** the loss decreases rapidly and smoothly. Here, the optimizer follows the geometry of the loss surface efficiently, making steady progress toward the minimum without oscillations. This typically yields the most effective and consistent convergence behavior.
- **Learning rate too large:** the loss becomes unstable, oscillates, or diverges. The update steps overshoot the minimum, causing the optimizer to bounce across the loss landscape or move away from the optimal region entirely, preventing successful training.

To find a good learning rate, the LR-finder method [4] is used. The idea is to train the model for 1 epoch while increasing the learning rate exponentially from a very small value to a large value, and record the training loss.

In this project were tested two different sizes of hidden layer (64, 128) as shown in the "Fig. 6" & Fig. 7"

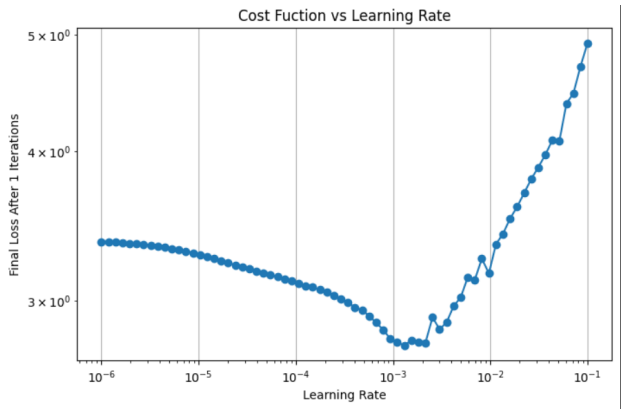

Fig. 6. Training Loss vs Learning Rate for hidden size = 64

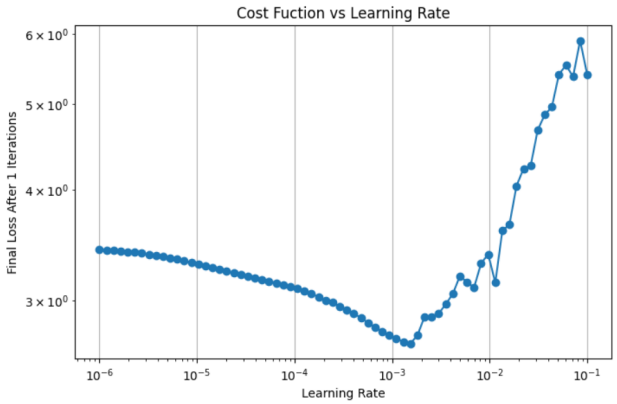

Fig. 7. Training Loss vs Learning Rate for hidden size = 128

The recommended learning rate is typically chosen at the point where the loss decreases most rapidly, or slightly before the loss begins to rise. Based on the observations from "Fig. 6", the optimal learning rate is approximately 1×10^{-3}, whereas for "Fig. 7" the optimal value is approximately 2×10^{-3}.

3) Regularization Factor (λ): L2 regularization, controlled by the parameter λ, helps prevent overfitting by penalizing large weight values. To determine ans appropriate value, the following range was evaluated:

Results: Below is a table that demostrates the results for a size of hidden layer 64 and a learning rate of 1×10^{-3}.

TABLE II
PERFORMANCE ACROSS DIFFERENT REGULARIZATION STRENGTHS.

λ	Val Acc	Val F1	Val Loss	Iter
0	0.99891	0.99891	0.04663	91
10^{-4}	0.99927	0.99927	0.04217	91
10^{-3}	0.99745	0.99745	0.06699	80
10^{-2}	0.99818	0.99818	0.06443	80
0.1	0.99782	0.99782	0.08300	105

Conclusion: The optimal regularization strength was found to be $\lambda = 10^{-4}$, which provides the best balance between generalization performance, predictive confidence, and training efficiency.

REFERENCES

[1] V. K. Dasari, "Multilayer perceptron and its hyperparameter tuning," https://medium.com/@vijaikdasari/multilayer-perceptron-and-its-hyperparameter-tuning-deedfbca2d1b, 2020, accessed: 2025-02-12.
[2] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[3] L. Y. Data, "Gradient descent algorithm: Key concepts and uses," https://labelyourdata.com/articles/gradient-descent-algorithm, 2024, accessed: 2025-11-19.
[4] S. Gugger, "How do you find a good learning rate?" https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html, 2018, accessed: 2025-02-12.