Artificial Neural Networks for Handwritten Digit Recognition

Minh An Nguyen

Swinburne University of Technology
School of Science, Computing and Engineering Technologies
Hawthorn, Victoria 3122, Australia
*Email: minhan6559@gmail.com

Abstract

Artificial Neural Network (ANN) is a powerful machine learning approach inspired by the structure of biological neurons. This report provides a foundational overview of Multilayer Perceptron (MLP) - a foundational type of ANN and their application to handwritten digit recognition. Core MLP components, including neurons, layers, and activation functions, are discussed, along with the training process that enables pattern recognition and prediction. The challenges in handwritten digit recognition are addressed, highlighting the effectiveness and limitations of MLPs in overcoming these obstacles.

Key words: artificial intelligence, neural network, digit recognition, deep learning, multilayer perceptron

I. Introduction

Artificial Neural Networks (ANNs) have emerged as a powerful algorithm in machine learning, mirroring the functionality of biological neurons. Inspired by the human brain's ability to learn and recognize patterns, ANNs have found diverse applications, ranging from image and speech recognition to natural language processing. One such captivating application lies in the realm of handwritten digit recognition, a task that has long challenged traditional algorithmic approaches due to the variability in human handwriting styles.

This report will explore the foundational building blocks of a basic architecture of ANN - the Multilayer Perceptrons (MLPs), including neurons, layers, and activation functions, and illuminate the iterative training process that empowers these networks to learn and generalize from data.

A key focus of this report is the application of MLPs to the MNIST dataset, a widely recognized benchmark for handwritten digit recognition. This dataset comprises thousands of labeled images of handwritten digits, providing a rich and diverse training ground for ANN models.

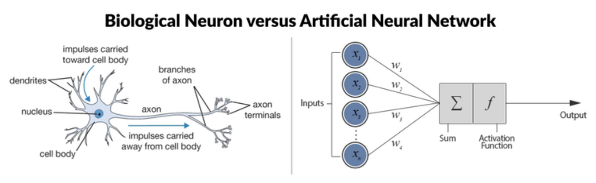


Fig. 1. Biological Neuron versus Artificial Neural Network [3]

II. Architecture of a basic neural network

Artificial Neural Networks (ANNs) comprise diverse architectures, each tailored for specific tasks. This report focuses on the fundamental building block: the Multilayer Perceptron (MLP). As a feedforward network, the MLP processes information in a unidirectional flow, making it a suitable starting point for understanding ANN principles.

II.1. Neuron

Each neuron in an MLP is like a simple unit, that performs basic mathematical operations on its inputs. It receives a set of input values, represented as a vector $\mathbf{x}$, which could correspond to features of a data sample. These inputs interact with the neuron's unique parameters: a weight vector $\mathbf{w}$ and a bias term b. The neuron performs a weighted sum of the input values, essentially calculating a weighted average based on its current parameters. To this sum, the bias term is added, providing an adjustable offset. Finally, the result of this computation is passed through an activation function.

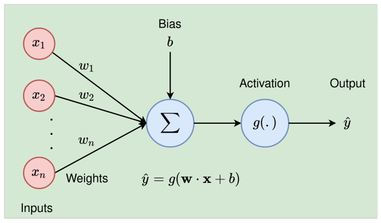


Fig. 2. Single Neuron [6]

II.2. Perceptron

Introduced by Frank Rosenblatt in 1957, the perceptron stands as the most fundamental neural network structure [9]. It comprises multiple input values (corresponding to data features), a single neuron for processing, and a solitary output. The perceptron's operation hinges on forward propagation, a process involving three distinct steps:

1. **Weighted Summation:** Each input value (x_i) is multiplied by its respective weight (w_i), reflecting the strength of the connection between the input and the neuron. The products are then summed, effectively calculating the dot product of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$.

$$z = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i$$

2. **Bias Addition:** A bias term, an adjustable constant, is added to the weighted sum. This bias provides flexibility in shifting the decision boundary of the neuron.

$$z := z + b$$

3. **Activation:** The combined result of the weighted sum and bias is passed through an activation function. This non-linear function introduces complexity and enables the perceptron to learn non-linear relationships between inputs and outputs.

$$\hat{y} = \sigma(z)$$

II.3. Layer

A Multilayer Perceptron (MLP) is organized into distinct layers of interconnected neurons, each serving a specific role in the flow of information:

1. **Input Layer:** The input layer acts as the gateway for data into the network. Each neuron in this layer corresponds to a single feature or input variable. These neurons simply pass the input values onto the next layer without performing any computation. The number of neurons in the input layer directly matches the number of features in your dataset. For the MNIST dataset, each handwritten digit is represented as a 28x28 pixel grayscale image, so the input layer of the MLP would consist of 784 neurons (28 * 28), each corresponding to a single pixel value ranging from 0 (black) to 255 (white).
2. **Hidden layers:** positioned between input and output, are the computational core of an MLP. Each hidden layer contains multiple neurons that process weighted inputs, apply biases, and utilize activation functions to generate outputs. The outputs of one hidden layer become the inputs for the next, allowing the network to model complex relationships in the data.
3. **Output Layer:** The output layer is responsible for producing the final classification decision for a given input digit. In the case of MNIST, since there are 10 possible digit classes (0 to 9), the output layer typically consists of 10 neurons. Each output neuron corresponds to one of the digit classes and produces a probability score indicating the network's confidence in that particular classification. The digit with the highest probability score is deemed the network's prediction for the input image. The activation function in the output layer is usually a softmax function.

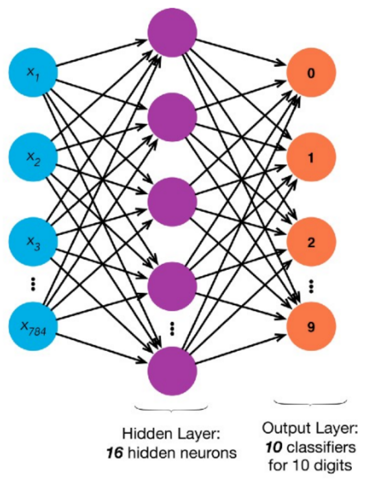

x_1
x_2
x_3
x_{784}
0
1
2
9
Hidden Layer: 16 hidden neurons
Output Layer: 10 classifiers for 10 digits

Fig. 3. Layers [5]

II.4. Activation Functions

Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns in data. Here are four common activation functions used in MLPs, along with their mathematical definitions and derivatives:

ReLU (Rectified Linear Unit):

- Function: $f(x) = \max(0, x)$
- Derivative: $f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

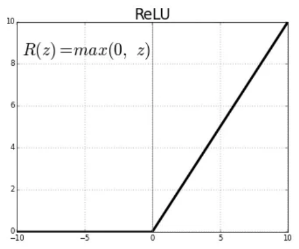


Fig. 4. ReLU

Sigmoid:

- Function: $f(x) = \frac{1}{1+e^{-x}}$
- Derivative: $f'(x) = f(x) * (1 - f(x))$

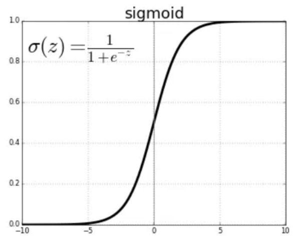


Fig. 5. Sigmoid

Softplus:

- Function: $f(x) = \ln(1 + e^x)$
- Derivative: $f'(x) = \frac{1}{1+e^{-x}}$

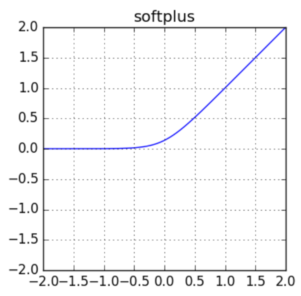


Fig. 6. Softplus

tanh (Hyperbolic Tangent):

- Function: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Derivative: $f'(x) = 1 - (f(x))^2$

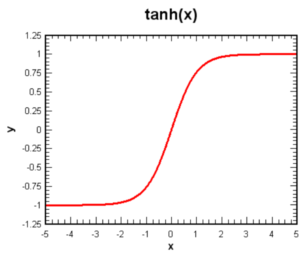


Fig. 7. tanh

In multi-class classification tasks, where a model needs to choose among multiple categories, the `softmax` function is the go-to activation for the output layer. It transforms the raw output values from the network into probabilities, ensuring they sum to one and represent a valid probability distribution across all classes.

Softmax (Output Layer):

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where:

- K: the total number of classes.
- i: the index of each class in the input vector.

III. Learning Process

III.1. Vectorization

Each layer in an MLP involves repetitive computations that can be optimized using vectorization. By transforming weight vectors into a weight matrix W and bias terms into a vector b, we can express layer calculations as a single matrix equation, boosting efficiency. This matrix representation also simplifies the network's mathematical expression.

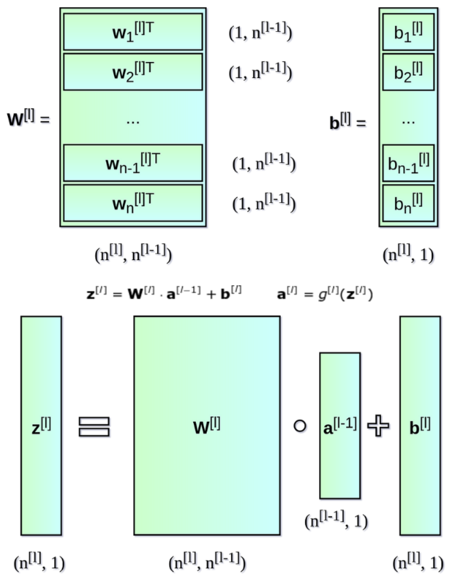


Fig. 8. Vectorization [2]

III.2. Forward Propagation

In a multilayer perceptron (MLP) processing a dataset with m samples, each containing nx features, forward propagation involves the efficient transformation of data through the network's layers.

Each sample's feature values are organized into an input vector x of size nx, and these vectors are stacked vertically to form the input matrix X. Each layer's neuron outputs are similarly arranged into a vertical vector a, which are further stacked to form the activation matrix A.

For a layer l, the forward propagation involves two steps:

1. **Linear Transformation:** The input activation $A^{[l-1]}$ from the previous layer is multiplied by the weight matrix $W^{[l]}$ and a bias vector $b^{[l]}$ is added.

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

2. **Activation:** The activation function g is applied element-wise to the matrix $Z^{[l]}$, resulting in the activation $A^{[l]}$ of layer l:

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

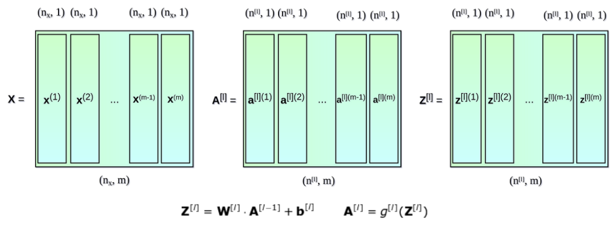

$X =$
$x^{(1)}$ $x^{(2)}$ ... $x^{(m-1)}$ $x^{(m)}$
$(n_x, 1)$
(n_x, m)
$A^{[l]} =$
$a^{[l](1)}$ $a^{[l](2)}$ ... $a^{[l](m-1)}$ $a^{[l](m)}$
$(n^{[l]}, 1)$
$(n^{[l]}, m)$
$Z^{[l]} =$
$z^{[l](1)}$ $z^{[l](2)}$ ... $z^{[l](m-1)}$ $z^{[l](m)}$
$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$
$A^{[l]} = g^{[l]}(Z^{[l]})$

Fig. 9. Forward Propagation [2]

III.3. Loss Function

The learning progress of a neural network is assessed by a loss function, which indicates the difference between predicted and actual outputs. For multi-class classification problems like handwritten digit recognition, a commonly used loss function is categorical cross-entropy. This function measures the difference between the predicted probabilities and the true labels, guiding the network's learning process toward accurate predictions.

The categorical cross-entropy loss function for a single training example is defined as:

$$L = -\sum_i y_i \log(\hat{y}_i)$$

In practice, for a batch of m training examples, the average loss is computed as a cost function:

$$J = -\frac{1}{m}\sum_{j=1}^{m}\sum_i y_{ji}\log(\hat{y}_{ji})$$

where:

- L is the loss for a single training example.
- J is the average loss across the batch.
- m is the number of training examples in the batch.
- y_i (or y_{ji}) is the true probability for class i (1 for the correct class, 0 for other classes).
- $\hat{y}_i$ (or $\hat{y}_{ji}$) is the predicted probability for class i, typically obtained from the softmax activation in the output layer.

III.4. Backward Propagation

Backpropagation is the backbone of neural network training. It is an algorithm that calculates the gradients of the loss function for each weight and bias in the network, allowing for iterative updates that improve the model's accuracy.
Backpropagation begins at the output layer. It utilizes the chain rule of calculus to compute the gradients of the loss function for the weights and biases in each layer, moving backward from the output layer to the input layer.

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

$$db^{[l]} = \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$$

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

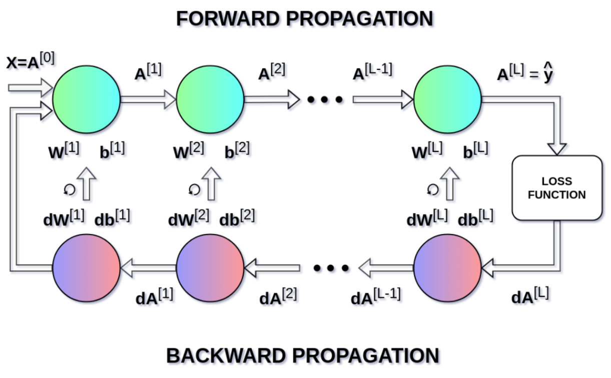


Fig. 10. Backward Propagation [2]

III.5. Gradient Descent

Neural networks learn by adjusting their internal parameters (weights and biases) to minimize a cost function that measures their errors. This optimization is achieved using gradient descent, a method that calculates the direction of the steepest descent on the cost function's surface. The network gradually improves its predictions by iteratively changing the parameters in this direction, aligning them more closely with the lowest cost value.

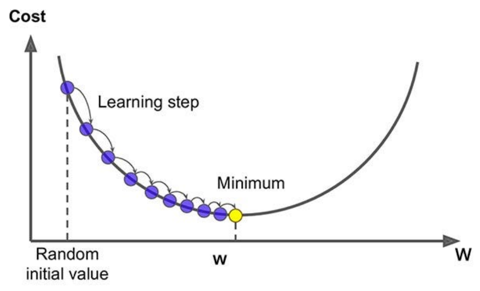


Fig. 11. Learning with Neural Network [8]

This gradient descent process is guided by the gradients of the loss of each parameter, calculated through backpropagation. The update rules for the parameters can be expressed as:

$$W^{[l]} := W^{[l]} - \alpha\frac{\partial \mathcal{L}}{\partial W^{[l]}}$$

$$b^{[l]} := b^{[l]} - \alpha\frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

α (alpha) is the learning rate, a hyperparameter that determines the step size of each update. If the learning rate is too small, the network will learn slowly and may get stuck in suboptimal solutions. Conversely, if the learning rate is too large, the optimization process may overshoot the optimal solution and fail to converge. Choosing the appropriate learning rate is essential for efficient and effective training of neural networks.

III.6. Overview

The learning process in a multilayer perceptron (MLP) is described as below:

1. **Initialization:** The MLP's weights and biases are initialized with specific strategies.
2. **Forward Propagation:** Input data is fed forward through the network, layer by layer, with each neuron transforming its weighted input and applying an activation function.
3. **Loss Calculation:** The predicted output is compared to the ground truth using a loss function, such as categorical cross-entropy.
4. **Backpropagation:** Starting from the output layer, the gradients of the loss function for each weight and bias are calculated.
5. **Parameter Update:** An optimization algorithm, such as gradient descent, utilizes the gradients to update the weights and biases. The learning rate, a hyperparameter, controls the step size of these updates.
6. **Iteration:** Steps 2-5 are repeated for multiple epochs until the loss converges or a stopping criterion is met.

IV. Handwritten Digit Classification with Multilayer Perceptrons

In this section, we'll explore how multilayer perceptrons (MLPs), a type of artificial neural network, can be used to recognize handwritten digits.

To make this exploration interactive, "ANNalyze" - a Ruby application has been developed, which can be found at this Github repository. This application enables users to not only create and train their own MLP models but also to test these models by providing their own handwritten digits as input. By combining theoretical understanding with hands-on experimentation, users can gain a deeper understanding for the capabilities and limitations of MLPs in the context of handwriting recognition.

The application, named "ANNalyze", is a clever combination of "ANN" (Artificial Neural Network) and "analyze", reflecting its core functionalities. Interestingly, the name also aligns with the first name of the app's creator, "An", adding a personal touch to this innovative tool.

"ANNalyze" utilizes two Ruby gem libraries for its implementation: **Numo::NArray** for efficient numerical computation and linear algebra, and **Ruby2D** for creating graphical user interfaces.

IV.1. Dataset: MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely-used benchmark for evaluating handwritten digit recognition models [4]. It comprises a collection of grayscale images, each depicting a single handwritten digit (0-9) centered within a 28x28 pixel grid. Each pixel's intensity is represented by a value ranging from 0 (black) to 255 (white), translating into 784 input features for the MLP. The dataset is divided into two distinct sets:

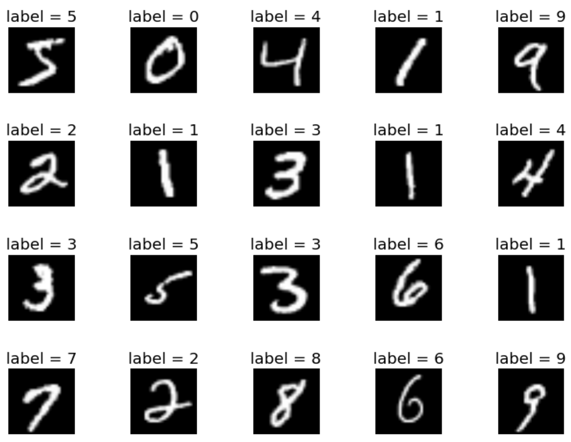


Fig. 12. MNIST dataset

Training Set: This set contains 10,000 images used for training the MLP model.

Test Set: This set includes 1,000 images used to evaluate the model's performance after training.

Table 1. Distribution of MNIST Labels

Label	Frequency
0	969
1	1145
2	1058
3	979
4	1025
5	883
6	950
7	1017
8	985
9	989

IV.2. Building The Model

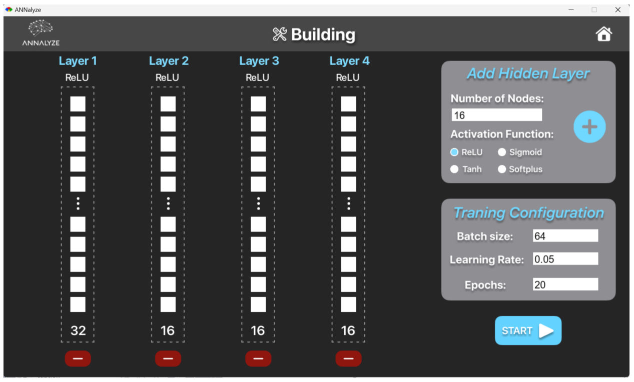
Users can design an MLP architecture comprising up to four hidden layers with the output is a softmax with 10 nodes. For each hidden layer, they can independently specify:

- **Number of Nodes:** This determines the number of neurons within that layer.
- **Activation Function:** Users can select from a variety of activation functions, including ReLU, Sigmoid, Tanh, and Softplus.

Below the layer configuration, users can fine-tune the training process by adjusting:

- **Batch Size:** The number of training samples used in each iteration of gradient descent.
- **Learning Rate:** A hyperparameter that controls the step size of updates.
- **Epochs:** The number of times the entire training dataset is passed through the network.

Once these parameters are configured to the user's satisfaction, clicking the "Start" button initiates the training process.



Number of Nodes: 16
Activation Function: [✓ ReLU] Sigmoid Tanh Softplus
Batch size: 64
Learning Rate: 0.05
Epochs: 20

Fig. 13. Building Screen

IV.3. Training The Model

The "Training" screen within ANNalyze provides a dynamic view of the model's learning progress:

- **Progress Bar:** Visually tracks the percentage of completed epochs.
- **Epoch:** Displays the current epoch number.
- **Cost:** Shows the average loss across the current training batch.

- **Accuracy:** Calculates and displays the accuracy on the test set after each epoch.

Upon completion of training:

- **Final Accuracy:** The final accuracy achieved on the test set is presented.
- **Saving:** Users can save their trained model with a custom name for future use.

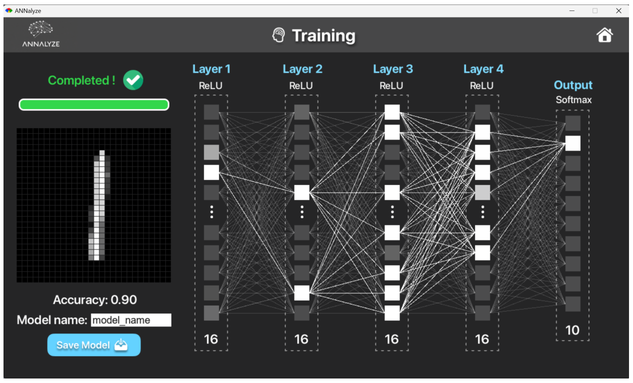


Fig. 14. Training Screen

IV.4. Inference Screen

The Inference screen in ANNalyze enables users to test their trained MLP models on new handwritten digits. After selecting a saved model, users can draw a digit on the canvas, then click "Predict" to see the model's classification. The screen also visualizes the model's architecture and the input digit for a transparent prediction process.

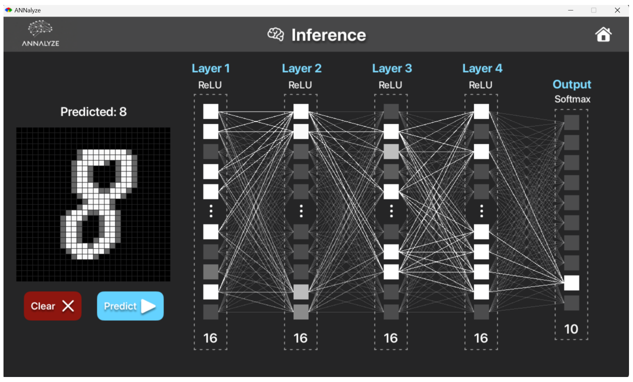


Fig. 15. Inference Screen

IV.5. Evaluating

To evaluate the effectiveness of MLP models created using ANNalyze for handwritten digit recognition, a set of MLP models was constructed and trained within the application.

Here's a table showcasing the performance of several MLPs with different architectures, all trained with ReLU activation on all hidden layers and 50 epochs on the MNIST dataset:

Table 2. MLP Performance on MNIST

Name	Number of nodes in each hidden layer	Learning Rate	Final Accuracy
MLP-1	64	0.01	89%
MLP-2	128	0.01	90%
MLP-3	64, 32	0.05	90.2%
MLP-4	128, 64	0.001	90.6%
MLP-5	256, 128, 64	0.005	91.1%

IV.6. Discussion

Multilayer Perceptrons (MLPs) offer a straightforward and versatile approach to various machine learning tasks, including image classification. However, when it comes to tasks involving image data, such as handwritten digit recognition, MLPs have notable limitations.

- **Lack of Spatial Awareness:** Unable to inherently capture relationships between neighboring pixels, which is crucial for understanding image content.
- **Need for Manual Feature Engineering:** Requires preprocessing steps to extract relevant features from images, adding complexity and potential for error.
- **Potential for Overfitting:** Large number of parameters can lead to overfitting, especially with limited training data.

Instead of MLP, there are some better approach for this problem like Convolutional Neural Networks (CNNs). CNNs are specialized neural networks designed to excel at image-related tasks. Their unique architecture, which incorporates convolutional and pooling layers, allows them to automatically learn spatial hierarchies of features directly from raw pixel data. This makes CNNs a more powerful and efficient choice for image classification, including handwritten digit recognition.

V. Summary

This report provides the fundamental concepts of multilayer perceptrons (MLPs), a foundational type of artificial neural network, and their application to handwritten digit recognition. The report further illustrated the practical use of MLPs through the ANNalyze application, highlighting its potential for users to experiment with different architectures and understand the impact of hyperparameter choices on model performance.

While not aiming to find the most optimal MLP for digit classification, this report provides a solid understanding of the mathematical principles behind MLPs and their real-world applications.

VI. Acknowledgement

I would like to express my sincere gratitude to Andrew Ng for his exceptional Deep Learning course, which provided the foundation for this report [7]. I am also grateful to BachNgoH, whose work inspired the development of the ANNalyze application [1].

References

1. BachNgoH. *CS163-NeuralVis*. GitHub, 2023.
2. Miloš Bai. *Let's code a Neural Network from scratch in Python (NumPy only)*. 2019.
3. DataCamp. *Deep Learning in Python*. 2024.
4. DARIEL DATO-ON. *MNIST in CSV*. Kaggle, 2017.
5. dudeperf3ct. *Force of Multi-Layer Perceptron*. October 2018.
6. Jose Fumo. *The Basics of Neural Networks — Neural Network Series - Part 1*. 2017.
7. Andrew Ng. *Deep Learning Specialization*. 2023.
8. Rishi. *Understanding of Gradient Descent: Intuition and Implementation*. 2023.
9. Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.