Department of Artificial Intelligence and Human Interfaces
University of Salzburg

PS Natural Computation
SS 24

Dynamic Optimization of Target Values in Neural Network Classification

July 2, 2025

Project Members:
Andrassik Leon, 12209906, leon.andrassik@stud.plus.ac.at
Bhuiyan Amreen, 12203597, amreen.bhuiyan@stud.plus.ac.at
Yakar Betül, 12205751, betuel.yakar@stud.plus.ac.at
Zauner Sofia, 12118492, sofia.zauner@stud.plus.ac.at

Academic Supervisor:
Helmut MAYER
helmut@cosy.sbg.ac.at

Correspondence to:
Universität Salzburg
Fachbereich AIHI
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Abstract

The aim of our project is to implement an alternative method of choosing target values for classification with neural networks. Instead of using traditional one-hot encoding, which uses fixed values of 0 and 1, this method involves assigning custom target values to each class.

These target values are dynamically optimized during the training process, which distinguishes our approach from conventional fixed-target training. Our idea is to capture the model's natural predictions and use these to refine the target values.

This approach requires a new rule for determining the predicted class: rather than choosing the neuron with the highest value, we select the neuron whose output value is closest to its corresponding target value and therefore has the lowest error; a method we will refer to as "minimum distance classification".

The goal of our project is to increase training efficiency while maintaining acceptable classification accuracy. To evaluate the performance of our approach, we will test it on the MNIST dataset and compare it with the one-hot encoding method.

1 Introduction

Artificial neural networks (ANNs) have become essential tools in modern classification tasks, mapping complex input data to discrete predefined output categories. During training, the error is calculated as the difference between the desired output and the actual output produced by the network. The objective of the training process is to minimize this error through iterative adjustments of the network's weights and biases. Following the training phase, performance is evaluated using a test set that contains previously unseen data, providing a neutral measure of its generalizability and classification accuracy.

Traditionally, the learning process of such networks is based on one-hot encoded target values, where the correct class is represented by the value 1, and all others are assigned a 0. This binary approach became standard due to its simplicity and compatibility with widely used loss functions. However, enforcing these extreme target values could potentially lead to challenges depending on the system's underlying activation function. It may negatively affect gradient flow, reduce training efficiency, and encourage overconfident predictions.

One-hot encoding enforces a "winner-takes-all" (WTA) dynamic, where only the neuron with the highest output is considered correct. Since the target vector consists of a 1 for the correct class and a 0 everywhere else, this is effectively equivalent to selecting the output that is closest to its respective target value. Thus, WTA can be seen as a special case of a more general strategy, which is known as nearest target or minimum distance classification. Here we make a general comparison between the entire output distribution given by the network and the available target vectors for each class. Whichever class's target the output is closest to (using a distance metric such as Euclidean distance) will be chosen as the correct class.

In this project, we propose a more flexible approach for target value assignment in classification networks. Instead of defining fixed values of 0 and 1, we generate target vectors composed of custom class and non-class values. Here, class values refer to the target values assigned to the positions which would ordinarily be filled with 1 in a one-hot vector, while the non-class values fill all other positions. These alternative targets are not chosen arbitrarily, but rather inferred from the model's own inherent output tendencies during training. By adapting target values dynamically in response to the network's evolving behavior, we aim to guide learning in a way that is more aligned with the model's natural predictions.

This change also calls for a rethinking of how predictions are interpreted. Instead of relying on the activation of the highest neuron to determine the predicted class as in WTA, we introduce a "nearest target classification" approach. We select the class whose output vector value is closest to the corresponding target vector, thereby minimizing the error. This nearest target classification strategy reflects the idea that a well-trained model should not necessarily produce a maximal value in any particular output neuron, but rather accurately match the overall desired response.

Our hypothesis is that this method improves the training efficiency, while maintaining the classification accuracy. We empirically evaluate this method using the MNIST dataset, comparing its performance against the traditional one-hot encoding approach.

2 Classification with Artificial Neural Networks

3 Methodology

In this section, we describe our approach to implementing dynamic target value optimization for neural network classification. First, we break down the concrete architecture of the target vectors used for training and classification. We then discuss how the class and non-class values that make up these target vectors are initially constructed and subsequently optimized dynamically during training.

3.1 Target Value Architecture

To establish our methodology clearly, we first review the traditional one-hot encoding approach before defining our alternative target value structure.

3.1.1 Traditional One-Hot Encoding

In standard classification with C classes, the target vector $\mathbf{t}^{(i)}$ for a sample belonging to class i is defined as:

$$\mathbf{t}^{(i)} = [0, 0, \ldots, 0, \underbrace{1}_{i\text{-th position}}, 0, \ldots, 0]$$

For example, with 3 classes, the target vectors are:

$$\mathbf{t}^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{t}^{(2)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{t}^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

3.1.2 Dynamic Target Value Structure

Our approach replaces the fixed values 0 and 1 with adaptive class and non-class values. For a classification problem with C classes, we define:

- c_i: the class value for class i (replaces 1 in one-hot encoding)
- $\bar{c}_i$: the non-class value for class i (replaces 0 in one-hot encoding)

The target vector $\mathbf{t}^{(i)}$ for class i becomes:

$$\mathbf{t}^{(i)} = [\bar{c}_i, \bar{c}_i, \ldots, \bar{c}_i, \underbrace{c_i}_{i\text{-th position}}, \bar{c}_i, \ldots, \bar{c}_i]$$

Using the same 3-class example:

$$\mathbf{t}^{(1)} = \begin{bmatrix} c_1 \\ \bar{c}_1 \\ \bar{c}_1 \end{bmatrix}, \quad \mathbf{t}^{(2)} = \begin{bmatrix} \bar{c}_2 \\ c_2 \\ \bar{c}_2 \end{bmatrix}, \quad \mathbf{t}^{(3)} = \begin{bmatrix} \bar{c}_3 \\ \bar{c}_3 \\ c_3 \end{bmatrix}$$

Importantly, in our implementation:

- Each class has its own class and non-class values c_i and $\bar{c}_i$
- All non-class values within a class are the same
- All non-class values $\bar{c}_i$ (and in special cases class values) are optimized dynamically during training

3.2 Target Value Initialization and Optimization

Since a sample's classification depends on finding the nearest target vector to the network's output, we need target vectors that are reasonably well-separated from one another. Given our construction method, this separation can be achieved by ensuring that each class's class and non-class values are sufficiently spaced apart.

Our approach accomplishes this through two stages: first, we select initial class and non-class values based on the network's natural output tendencies, and second, we dynamically increase the spacing between these values during training. This section describes both processes.

3.3 Initial Target Value Selection

The key challenge in selecting the initial class and non-class values is balancing two competing requirements: the values should reflect the model's innate output tendencies rather than being chosen arbitrarily, while also providing a distribution that enables sufficient separation, either initially or through the subsequent optimization process.

We explored several strategies to achieve this balance:

1. Uniform distribution of class and non-class values

2. Traditional soft targets (e.g.: 0.8 and 0.2)
3. Mean of untrained-network predictions
4. Remapping the untrained output means to use the full $(0, 1)$ range

After experimenting with these approaches we discovered that these largely lead to predictions which are no better than random guessing, with the notable exception of the second approach, which we will cover more in the results section. Through further discussion and experimentation, we settled on an altered version of the third approach.

This approach closely resembled the third approach, but substitutes it by slightly nudging the resulting class and non-class values to increase the initial separation.

Algorithm 1 Adaptive Target Value Computation

Require: Network $\mathcal{N}$, test data $\mathcal{D}_{test}$, spacing function, nudge value ϵ
Ensure: Adjusted non-class and class threshold values for each class

// Compute mean activations for each class
for each $(data, target)$ in $\mathcal{D}_{test}$ **do**
  $output \leftarrow \mathcal{N}(data)$
  Group outputs by class: add to $outputs[target]$
end for
Extract class and non-class activation values: $(nc\_vals, c\_vals) \leftarrow \text{partition}(outputs)$
for each class c **do**
  $c\_means[c] \leftarrow \text{mean}(c\_vals[c])$
  $nc\_means[c] \leftarrow \text{mean}(nc\_vals[c])$
end for
// Apply spacing function to determine initial thresholds
$(non\_class\_values, class\_values) \leftarrow \text{spacing}(c\_means, nc\_means)$
// Apply nudge adjustments for better separation
for each class c **do**
  if $non\_class\_values[c] > class\_values[c]$ **then**
    $non\_class\_values[c] \leftarrow non\_class\_values[c] + \epsilon$
  else
    $non\_class\_values[c] \leftarrow non\_class\_values[c] - \epsilon$
  end if
  // Enforce bounds [0,1] with compensatory adjustments
  if $non\_class\_values[c] \leq 0$ **then**
    $non\_class\_values[c] \leftarrow 0$
    $class\_values[c] \leftarrow \min(\epsilon, 1)$
  else if $non\_class\_values[c] \geq 1$ **then**
    $non\_class\_values[c] \leftarrow 1$
    $class\_values[c] \leftarrow \max(1 - \epsilon, 0)$
  end if
end for
return $(non\_class\_values, class\_values)$

Algorithm 1 begins by running a full test epoch through the network and storing

the activation outputs grouped by class. For each class i, the mean activation in that class's designated output neuron across all samples of class i becomes the class value c_i, while the non-class value $\bar{c}_i$ is computed as the mean activation across all other output neurons for samples of that same class.

This process typically results in densely packed threshold values that provide insufficient separation for effective classification. To address this, the method first applies the chosen spacing function (detailed in the next section) to establish initial thresholds, then further separates the class and non-class values by applying small nudge adjustments of magnitude ϵ. The nudging direction depends on the relative positioning of the values: if non-class values exceed class values, we increase the non-class threshold; otherwise, we decrease it. Boundary enforcement ensures that all values remain within $(0, 1)$ while maintaining meaningful separation.

3.4 Dynamic Target Optimization Algorithm

While the target values created by the algorithm 1 are, according to our criteria, based on the innate tendencies of the models and are somewhat well separated there are two issues with the targets generated by this approach.

1. As the network learns, the target values will quickly lose any relation to the networks innate output tendencies.
2. The value's separation depends greatly on the chosen ϵ. Small values of ϵ result in target values that rarely exceed a range of $(0, 0.3)$, likely because the underlying network outputs are compressed by the softmax function before averaging.

To address these limitations we continuously adapt these generated target values as training progresses, using a method we have come to call "sigma spacing".

Algorithm 2 implements this approach by replacing the fixed nudge parameter ϵ with a dynamic separation criterion based on the standard deviation of current network outputs.

Rather than relying on static target values computed from the initial untrained network, sigma spacing recomputes separation requirements after each training epoch using σ_{sum}, the sum of standard deviations across current outputs. This is meant to ensure that target values remain relevant as the network's behavior evolves during training. Additionally, by scaling adjustments according to the actual variability in network outputs, the method automatically adapts to the compressed range typical of softmax-normalized activations.

Algorithm 2 Standard Deviation-Based Threshold Adjustment

Require: Class values $class\_values$, non-class values non_class_values, scaling factor λ (default = 1)
Ensure: Adjusted threshold values with minimum separation guarantee

// Compute variability measure
$\sigma_{sum} \leftarrow$ sum of standard deviations across all current outputs
for each class c **do**
  // Check separation requirement
  if $|class\_values[c] - non\_class\_values[c]| < \sigma_{sum}$ **then**
    // Adjust non-class value based on relative positioning
    if $non\_class\_values[c] \geq class\_values[c]$ **then**
      $non\_class\_values[c] \leftarrow non\_class\_values[c] + \lambda\sigma_{sum}$
    else
      $non\_class\_values[c] \leftarrow non\_class\_values[c] - \lambda\sigma_{sum}$
    end if
    // Handle boundary violations with compensation
    if $non\_class\_values[c] > 1$ **then**
      $excess \leftarrow non\_class\_values[c] - 1$
      $non\_class\_values[c] \leftarrow 1$
      $class\_values[c] \leftarrow \max(0, class\_values[c] - excess)$
    else if $non\_class\_values[c] < 0$ **then**
      $deficit \leftarrow |non\_class\_values[c]|$
      $non\_class\_values[c] \leftarrow 0$
      $class\_values[c] \leftarrow \min(1, class\_values[c] + deficit)$
    end if
  end if
  // Final bounds enforcement
  $class\_values[c] \leftarrow \text{clip}(class\_values[c], 0, 1)$
  $non\_class\_values[c] \leftarrow \text{clip}(non\_class\_values[c], 0, 1)$
end for
return $(non\_class\_values, class\_values)$

4 Experimental Setup

Our experimental setup is designed to evaluate the performance of our dynamic target value optimization method against traditional one-hot encoding and soft-target baselines. All experiments are conducted on the MNIST dataset.

4.1 Dataset and Preprocessing

We used the standard MNIST dataset, which consists of 60,000 training images and 10,000 test images of handwritten digits (0-9). The images are normalized using a mean of 0.1307 and a standard deviation of 0.3081, which are the standard values for this dataset.

4.2 Network Architecture

All experiments use the same Convolutional Neural Network (CNN), implemented in PyTorch.

- An initial convolutional layer with 1 input channel, 10 output channels, and a kernel size of 5.
- A second convolutional layer with 10 input channels, 20 output channels, and a kernel size of 5. A dropout layer is applied after this convolution.
- A fully connected layer with 320 input features and 50 output features.
- A final fully connected layer with 50 input features and 10 output features, corresponding to the 10 digit classes.

The network uses Sigmoid activation functions after the convolutional and first fully connected layers, and a log-softmax activation function on the output layer.

4.3 Training Parameters

The training process is configured with the following hyperparameters:

- **Epochs:** The number of training epochs is configurable, but typically set to 4 for our experiments.
- **Batch Size:** We use a batch size of 64 for training and 1000 for testing.
- **Optimizer:** Stochastic Gradient Descent (SGD) is used as the optimizer, with a learning rate of 0.01 and momentum of 0.5.
- **Loss Function:** The loss is calculated using the Negative Log-Likelihood Loss (NLLLoss) between the network's log-softmax output and the target vectors.

4.4 Target Value Strategies

We compare three different target value strategies:

1. **One-Hot Encoding:** The traditional approach where the target vector for the correct class contains a 1 at the corresponding index and 0s elsewhere.
2. **Soft Targets:** A variation where the target for the correct class is set to 0.8 and the targets for all other classes are set to 0.2.
3. **Dynamic Targets:** Our proposed method, where the class and non-class target values are initialized and then dynamically adjusted during training using the "sigma spacing" adaptation method. The initial values are determined by the mean activations of the untrained network on the test set, with a small "nudge" of 0.2 applied for initial separation.

4.5 Evaluation Metrics

The performance of each strategy is evaluated based on the following metrics, which are logged using Weights & Biases:

- **Hard Accuracy:** The percentage of correctly classified images. For the dynamic target method, classification is determined by the nearest target vector, not the maximum output value.

- **Test Loss:** The average NLLLoss on the test set.
- **Confidence:** The average cosine similarity between the network's output and the target vector of the predicted class.
- **Target Value Separation:** For the dynamic target method, we track the absolute difference between the class and non-class target values for each class, as well as the average, minimum, and maximum separation across all classes.

5 Results and Analysis

6 Discussion and Potential Next Steps

7 Conclusion

8 Milestones

- **26.03.25 - Project Kickoff and Theoretical Outline**
  Create a rough outline of the project's content based on the given paper. Include loose definitions and a short summary of the introduced methods.
- **09.04.25 - Goal Specification**
  Clarify the project idea, define concrete project goals, and deepen the understanding of the topic. Focus especially on precise definitions of target values, class values, and non-class values. Write the abstract and milestone plan for the semester.
- **23.04.35 - Algorithm Design and Planning**
  Start working on the paper. Deepen research on alternative target value methods for classification with neural networks. Select and define an approach for alternative target encoding. Plan implementation steps for generating "class-" and "non-class" values.
- **07.05.25 - Initial Implementation and Testing of target value algorithm**
  Initial implementation of the new target encoding method. Before testing, allocate time for experimental design to ensure solid and interpretable results. Conduct simple experiments using simplified datasets to validate functionality.
- **21.05.25 - Compelete Implementation and Evaluate on MNIST-Experiments**
  Complete implementation and testing of the alternative target value methods. Prepare and conduct experiments on the MNIST dataset. Analyze model predictions to evaluate performance compared to traditional one-hot encoding. Discuss results and possible improvements.
- **04.06.25 - Optimization and Comparative Analysis**
  Complete improvements and experiments. Analyze experimental results and compare training efficiency and classification accuracy with traditional methods. Document findings and prepare initial draft of project report.
- **18.06.25 - Finalization**
  Final preparations for submission. Complete report and presentation.

9 Progress of Work

10 Subproject Responsibilities

- **Programming of Target Value Calculation Methods (Phyton)**
  Leon Andrassik
- **Research and Preparation of Target Encoding Methods**
  Betül Yakar, Sofia Zauner
- **Evaluation and Result Analysis**
  Amreen Bhuiyan, Betül Yakar, Sofia Zauner
- **Project Documentation and Preparation of Presentations**
  Leon Andrassik, Amreen Bhuiyan, Betül Yakar, Sofia Zauner
- **Presenting Progress Presentations**
  Leon Andrassik, Betül Yakar, Sofia Zauner
- **Presenting Final Presentation**
  Amreen Bhuyian

11 Links

- Project Page: `http://student.cosy.sbg.ac.at/???`
- PS Page: `http://www.cosy.sbg.ac.at/~helmut/Teaching/PSRobotik/`