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 10, 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 considered 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

ANNs are computational models inspired by the functioning and structure of the human brain, designed to simulate the way biological neurons process information. These networks serve as a powerful tool in complex pattern recognition and play a fundamental role in solving machine learning problems. In this section, we provide an overview of Artificial Neural Networks exploring their basic architecture and functionality, while also giving insight on their application in classification tasks.

## 2.1 Structure of Artificial Neural Networks

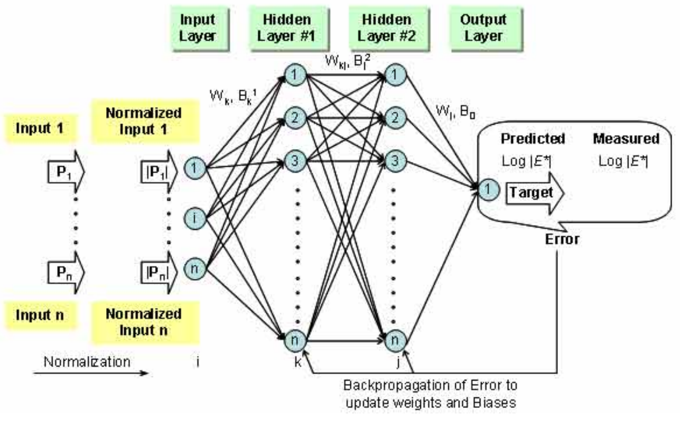


Figure 1: Network structure for training ANN models [1]

An Artificial Neural Network consists of a multitude of interconnected nodes, or neurons, organized into layers. Each layer is designed to process input data and pass the resulting output signals to the following. To achieve this, each neuron receives one or more inputs, applies respective weights and a bias, and then forwards the result through an activation function. Typically, a Neural Network is composed of three types of layers, as also visualized in Figure 1:

- **Input Layer:** The input layer is the first layer of the network and is responsible for receiving the raw input data. Each neuron in this layer represents a specific feature of the input. For example, in an image classification task each input neuron corresponds to a single pixel of the given image.
- **Hidden Layers:** The Hidden layers are located between the input and output layer. Most of the actual computation takes place here. The neurons in these layers take the output of the previous layer as input, process this information and send their output to the next layer. The number and size of the hidden layers vary depending on the complexity of the problem and the depth of the network.
- **Output Layer:** This concluding layer produces the networks's final output. In a classification task, the output layer usually consists of one neuron per possible class, with each neurons's output representing the network's predicition for the corresponding class.

Each neuron in a layer is connected to every neuron in the previous and the following layer, creating a fully connected, dense network. These connections are defined by the weights, which are adjusted during the training process.

## 2.2 Inner Workings of a Neuron

The process by which a neuron works can be described by the following two steps:

1. **Aggregation Function:** Each neuron receives one or more input signals. Within input layers these signals correspond to raw data, whereas in the case of hidden layers they originate from the outputs of the previous layer. Inter alia, the obtained data contains information on the corresponding weights and bias. The weights determine the strength of the connection between two neurons. During training the weights are adjusted accordingly, further explained in chapter 2.3 *Training Process.* The bias is used to shift the activation function to improve the learning process. Now the aggregation function, which is used to calculate the weighted sum of the given inputs by multiplying each input value by its corresponding weight, adding the bias and summing up the products, is applied. The weighted sum $z$ of a neuron can be computed as follows, where $x_i$ denotes the inputs, $w_i$ the weights and $b$ the bias:

   $$z_i = \sum_{i=1}^{n} x_i * w_i + b$$

2. **Activation Function:** In the next step, the weighted sum $z_i$ is passed through the activation function. This function introduces non-linearity into the network, which enables the system to recognize, learn and model complex non-linear patterns. The final output of the neuron in the layer is the result of the activation function applied to the weighted sum of its inputs. Two commonly used activation function, which were also applied in this project (see chapter 4.2 *Network Architecture*), are the Sigmoid- and the Softmax-activation function. They are defined as follows:

- **Sigmoid-activation function:** The Sigmoid function maps any input to a value between 0 and 1, making it ideal for binary classification problems.
  $\Rightarrow \sigma(z_i) = \frac{1}{1+e^{-z}}$
- **Softmax-activation function:** The Softmax function converts the raw output into a probability distribution across all k classes and is therefore often used in the output layer of classification networks.
  $\Rightarrow Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

## 2.3 Training Process

The objective of the training process in an ANN is to adjust the network's weights and biases to minimize the error between the actual output and the target values. This is achieved through a method called backpropagation. In this process the network iteratively updates its parameters using the gradients of the loss function with respect to each weight and bias in the direction that reduces the error. The loss function guides the optimization process by quantifying the difference between the predicted and the generated outputs. One commonly used loss function in classification tasks is the negative Log-Likelihood loss (NLL). This function measures the difference between the network's predicted probability distribution given by the log softmax function and the target vectors. The NLL is defined with $y_i$ denoting the target vector, $\hat{y}_i$ the generated proability distibution and M the number of classes as:

$$NLL(z_i) = -\sum_{j=1}^{M} y_i \; log \; \hat{y}_i$$

## 2.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a specialized type of Artificial Neural Network, designed for systems needing object detection, segmentation and recognition, such as image classification networks. The key difference between CNNs and traditional ANNs lies in their architecture. CNNs use convolutional layers to automatically detect sigificant patterns at different levels of abstraction like edges, textures or shapes, directly from the raw input data, eliminating the need for manual feature extraction. The architecture of a CNN typically consists of these three layer types:

- **Convolutional Layers:** These layers are the core components of CNNs. A convolutional layer applies filters, also known as kernels, to the input data performing a convolution operation that extracts the specific attributes. Each filter detects a different feature, like an edge or texture, in a small region of the input and generates a feature map that indicates the presence and strength of the corresponding feature within the input image. After convolution, the output is passed through an activation function, analog to the ANN structure.
- **Pooling Layers:** Pooling layers are typically placed after convolutional layers. They reduce the spatial dimensions of the feature maps by extracting the most important information, which also helps reducing the computational load. Common pooling methods include max-pooling and average-pooling.
- **Fully Connected Layers:** After several convolutional and pooling layers the network typically includes one or more fully connected layers. In these layers each neuron is connected to every neuron in the previous layer. They are

typically used to hierarchically combine the extracted features and make the final prediction or classification.

## 2.5 Classification Mechanisms

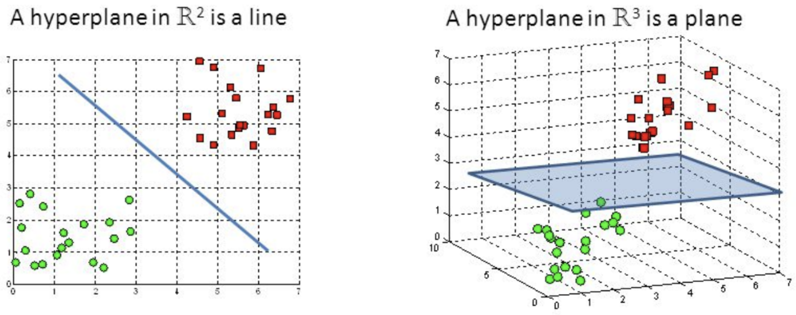


Figure 2: Illustration of hyperplanes in differnet dimensions [2]

In Artificial Neural Networks classification refers to the process of assigning given input values to one of several predefined classes based on specific patterns within the data. A key goal of a classification network is to find a decision boundary that can separate discrete regions corresponding to each class. ANNs find decision boundaries through an iterative process of training and adjusting the weights and biases accordingly to minimize classification error by using backporpagation. After the training phase, the network uses the learned decision boundary to classify unseen data points. The classification process can be broken down into three main steps: First, the input data is passed through the input layer, where each neuron represents a feature of the data. Next, the network processes the data through the hidden layers, where relevant features are extracted, using weighted sums, biases and activation functions. Finally, the processed data reaches the output layer, where the network assigns a class based on the neuron with the highest output value, providing the network's ultimate prediction.

To achieve the correct class assignment, the network tries to find a hyperplane, a line in a two-dimensional space or a plane in higher dimensions, that best separates the data points into distinct classes (illustrated in Figure 2). The decision boundary is typically a mathematical function, in which the input features are weighted, summed and passed through an activation function. If the output exceeds a certain threshold, the data point is classified as one class, otherwise it is classified into another.

However, when the data is not linearly separable, meaning a single straight line cannot separate the classes, the network uses multiple layers to create more complex non-linear decision boundaries. These non-linear boundaries are formed by stacking hidden layers and applying non-linear activation functions. Each layer learns progressively higher-level features of the data, allowing the network to model more complex patterns and therefore make more accurate desicions in classification tasks.

# 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.

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:

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

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 class and non-class values $c_i$ and $\bar{c}_i$
- There is only one non-class value per class, meaning each position which would be filled with 0 in onehot encoding is filled with this same non-class value $\bar{c}_i$ for that particular class's target $\mathbf{t}^{(i)}$
- 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, which we have decided to call "mean-nudge-initialization." 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 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 the 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 $\epsilon$. 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 network's innate output tendencies.
2. The value's separation depends greatly on the chosen $\epsilon$. Small values of $\epsilon$ 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 1** Adaptive Target Value Computation

```
Require: Network $\mathcal{N}$, test data $\mathcal{D}_{test}$, spacing function, nudge value $\epsilon$
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 the 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 2 implements this approach by replacing the fixed nudge parameter $\epsilon$ 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 $\sigma_{sum}$, 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 soft max-normalized activations.

**Algorithm 2** Standard Deviation-Based Threshold Adjustment

```
Require: Class values $class\_values$, non-class values $non\_class\_values$, scaling factor $\lambda$ (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 (NLL) 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 using mean-nudge initialization and then dynamically adjusted during training using the sigma spacing method. The initial values are determined by the mean activations of the untrained network on the test set, with a small "nudge" of 0.1 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 NLL 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

In this section, we will examine how the proposed method performs in practice with and without sigma spacing, how it compares to traditional methods like one-hot encoding and soft targets.

## 5.1 Target Values Over Time

Before looking at the actual performance of our approach, we will look at an example of how the target values are initialized and how they develop over time.

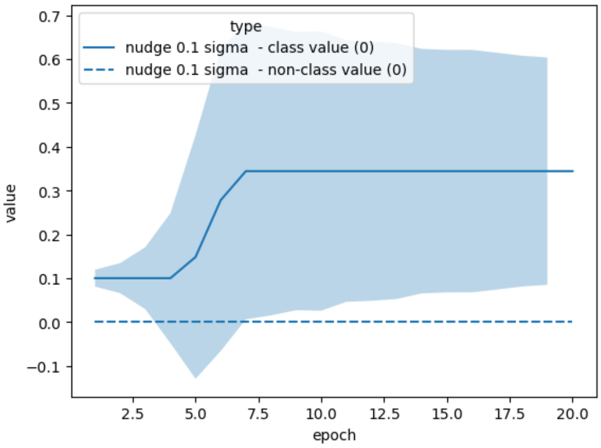


Figure 3: Class and non-class value development over 20 epochs

In Figure 3 see the class value (full) and non-class value (dotted) for class 0 in the MNIST dataset over the course of training the CNN. The blue area represents $\sigma_{sum}$, or rather, the area within a distance of $\sigma_{sum}$ from the class value.

The class and non-class values are likely initialized to around 0.1 and 0 respectively during pretraining. The nudge value $\epsilon = 0.1$ suggests that both were near 0 when initially generated and then nudged apart to their positions at the beginning of the graph.

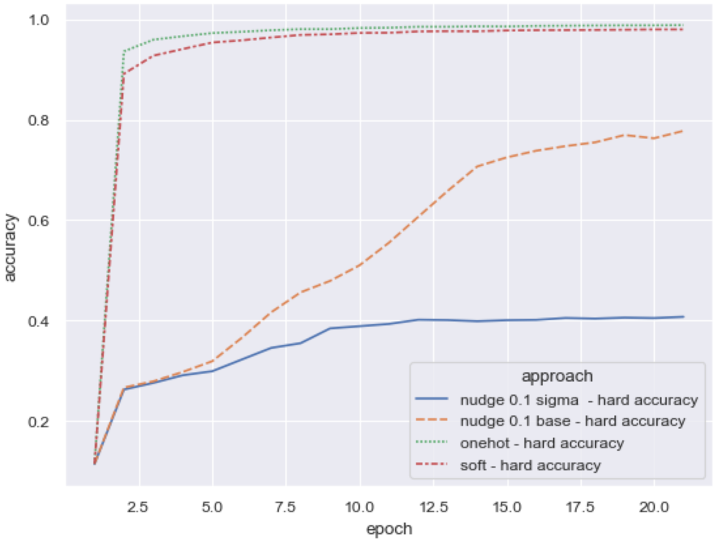


Figure 4: Accuracy Over Time

We also observe that, as described in Algorithm 2, when the non-class value comes within a distance of $\sigma_{sum}$ to the class value, we attempt to move the non-class value down. Since the non-class value is already at the lower bound of 0 here, the class value is moved up by $\sigma_{sum}$ instead.

## 5.2 Performance Comparison

To compare the performance of these approaches, we examine accuracy over training time, focusing on both final performance and convergence speed.

As shown in Figure 4, the one-hot and soft target approaches achieve over 90% accuracy within the first 3 epochs. In contrast, the mean-nudge initialized model without spacing reaches just under 80% accuracy after 20 epochs, while the variant using sigma spacing plateaus at approximately 40% accuracy by epoch 10.

## 5.3 Accuracy and Target Value Separation

In Figure 5 we compare the development of accuracy with the average distance between class and non-class values in each approach.

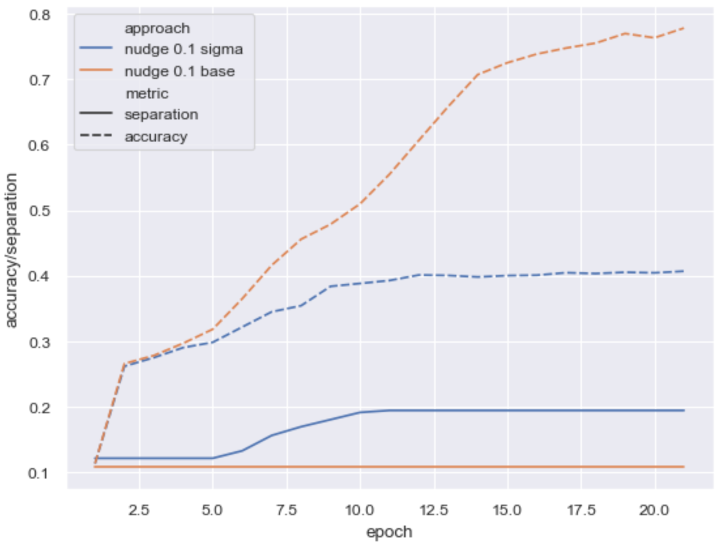
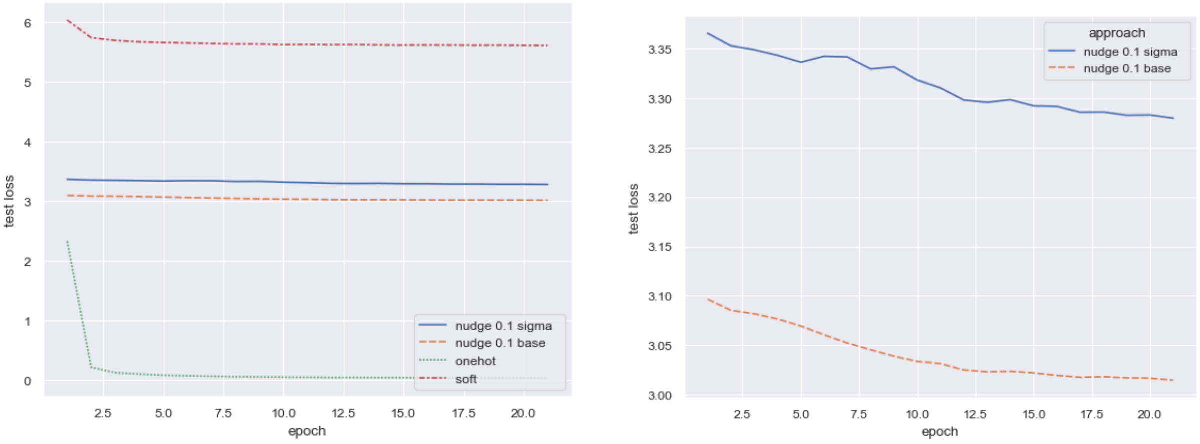


Figure 5: Accuracy vs. Target Value Separation

While the sigma spacing approach gradually increases its average separation between epochs 5 and 10, the approach without sigma spacing maintains a constant separation between the class and non-class values throughout training. The sigma-spaced approach enhances its accuracy only during the initial two epochs and during the separation change, while the non-spaced approach experiences an overall increase in accuracy throughout the training process.

## 5.4 Loss Development During Training

Figure 6 shows the development of training loss over the 20 epochs for all four approaches, as well as a magnified view of just the base and sigma spacing approaches.



(a) Loss development over training epochs

(b) Loss development of base and sigma approaches

Figure 6: Training loss comparison across different target value strategies

The test loss of the one-hot and soft target develop in line with their accuracy, showing a steep dropoff in the first few epochs, then slowly decreasing until the end of training. The loss in our proposed approaches decreases steadily throughout the entire training, without speeding up or slowing down significantly at any point.

## 5.5 Accuracy and Confidence

In Figure 7, we compare the final accuracy and confidence of the four different approaches after training for 20 epochs.

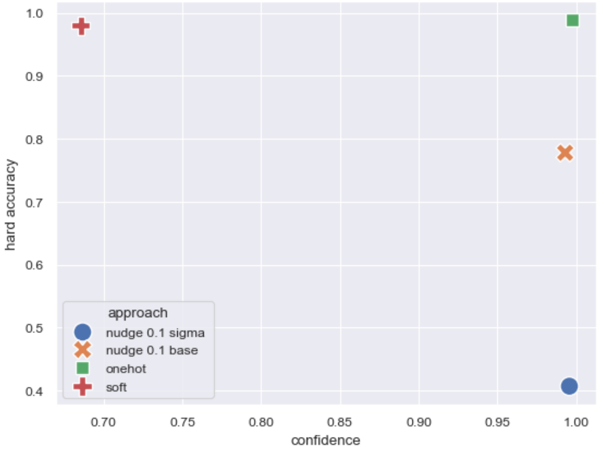


Figure 7: Accuracy vs. confidence

While accuracy varies dramatically across methods (100% for soft/one-hot targets, 80% for mean-nudge without spacing, 40% for sigma spacing), confidence remains consistently high ($\sim 100\%$) for all approaches except soft targets. This creates a stark mismatch between confidence and actual performance for the mean-nudge methods.

# 6 Discussion and Potential Next Steps

The aim of this section is to analyze the results of our experiments and find potential explanations for the observed behavior, as well as to point out any meaningful correlations in our data.

## 6.1 Performance Analysis: The Confidence-Accuracy Mismatch

The core insight which can be gained from these experiments is that these novel approaches are, in their current form, not very valuable and fail to fill a useful niche. The initial goal was to achieve better training speed or reduce overfitting while maintaining useful accuracy. However, our results demonstrate that the proposed dynamic target methods deliver neither benefit: they achieve significantly lower accuracy (40-80%) compared to traditional methods ($> 90\%$), while paradoxically maintaining high confidence ($\sim 100\%$) that undermines any potential regularization benefits (Figure 7). Where the one-hot approach delivers the best overall accuracy and training speed, the regular soft target approach exhibits similar accuracy with appropriately lower confidence, therefore acting against overfitting. Our dynamic approaches occupy the worst possible position in this trade-off space (poor accuracy combined with overconfident predictions) making them unsuitable for practical applications where either high performance or well-calibrated uncertainty is required.

## 6.2 The Divergence Between Loss and Accuracy

Where Figure 6(b) clearly shows the loss of our approaches decrease steadily during the entire training process, the accuracy (as seen in Figure 4), especially of the sigma spacing approach, does not mirror this trend as might be expected. While the decreasing loss does indicate that the network is learning, it appears that the sigma spacing inhibits the translation of this learning into improved classification accuracy. A potential explanation may be that, while the predictions do overall become more similar to the correct targets, the frequent movement of the class and non-class values prevents this increase in similarity from actually resulting in the closest target vector being the one from the correct class.

While this ”moving target” explanation could potentially contribute to the issue, the target values do eventually stabilize around epoch 10 (Figure 5). This indicates that there exists some more profound issue with (this interpretation of) the sigma spacing approach, which drives class and non-class values into positions which are fundamentally unsuitable for effective classification.

## 6.3 Target Value Dynamics and Their Impact

While the sigma spacing approach's low accuracy and high confidence provide little hope for the methods' usefulness, the median-nudge initialization approach (without spacing) exhibits a more salvageable accuracy / confidence ratio.

The concept of adapting the labels according to some external factor is not entirely novel, for example in adaptive label smoothing, where the smoothing factor is determined by a property of the input data [3]. Despite this, assigning labels solely based on the untrained network's output is an unexplored field that could use more study.

## 6.4 Implications for Future Work

There are a number of possible next steps that could be helpful in enhancing the efficacy of these methods. We will briefly outline possible areas of interest for further research in this area.

## 6.5 Mean-Nudge Initialization

The mean-nudge initialization pretraining approach, while being the most usable option out of our tested approaches, is still underperforming when compared to established techniques like one-hot encoded targets. We will suggest two potential improvements.

- Remapped-mean-nudge initialization: Since our current approach initializes the class and non-class values based on their observed means over the outputs for all samples of that particular class, our initial values fall only within a range that is typical of the softmax function applied to our model's output. As such, initial values have essentially no chance of exceeding $\sim 0.3$, despite the available range of values spanning the entirety of (0,1). It may be worth attempting to scale the initial class and non-class values to make more adequate use of the available range. While this could lead to better separated targets, it may also weaken the relation between the class and non-class values and the actual preferences of the network.

- Late initialization: While our proposed initialization method is applied entirely before the network is trained, it may be useful to generate these targets using a lightly trained network to extract more meaningful output preferences. The motivation is that even one or two epochs of training with traditional one-hot targets may encourage the network to form preferences more closely aligned with the classification task, though this may also introduce or reinforce unwanted biases.

## 6.6 Sigma Spacing

Since our implementation of sigma spacing had no positive impact on training, it may be useful to reevaluate some basic assumptions underlying our implementation. The way we have defined class and non-class values, while intuitive, may not be too abstract to accurately encode the network's actual preferences. Shifting from a one class value and one non-class value per class approach, to a one class and non-class value per neuron approach may or may not improve the effectiveness of the sigma spacing approach. Although this direction is speculative and lacks strong theoretical grounding at this stage, exploring such a reframing may still yield useful insights or unexpected improvements.

# 7 Summary and Conclusion

In this project, we explored an alternative method to one-hot encoding for initializing target values in neural network classification tasks, by assigning custom target values which depend on the models' untrained outputs and dynamically evolving them throughout training. This involved two main innovations:

- Mean-nudge initialization: inferring initial target values from the network's own output before training.
- Sigma spacing: adjusting these values during training based on the network's output variance.

## 7.1 Findings

Our main findings included an overall lack of viable performance across both our approaches when compared to traditional methods. Here the mean-nudge initialization method shows some promise if developed further, while the sigma spacing approach seems to suffer from several foundational issues which may require a rethinking of the underlying approach to be addressed adequately.

## 7.2 Conclusion

The idea of capturing the network's preferences remains interesting, but our methods may oversimplify the dynamics of the problem. While some parts of our approach seem to fall just short of being useful, other parts require more careful reconsideration.

Future works could focus on applying our existing sigma spacing approach to values generated using different assumptions about how to extract the ”preferences” of the neural network, or attempt to improve the initialization method itself by making better use of the available value interval (0,1).

# 8 Links

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

# References

[1] Federal Highway Administration. Ltpp computed parameter: Dynamic modulus, 2011.

[2] DeepAI. Hyperplane. Last accessed: 2025-07-10.

[3] Ujwal Krothapalli and A. Lynn Abbott. Adaptive Label Smoothing, December 2020. arXiv:2009.06432 [cs].