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 3, 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

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 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
1: **// Compute mean activations for each class**
2: **for** each $(data, target)$ in $\mathcal{D}_{test}$ **do**
3:     $output \leftarrow \mathcal{N}(data)$
4:     Group outputs by class: add to $outputs[target]$
5: **end for**
6: Extract class and non-class activation values: $(nc\_vals, c\_vals) \leftarrow \text{partition}(outputs)$
7: **for** each class c **do**
8:     $c\_means[c] \leftarrow \text{mean}(c\_vals[c])$
9:     $nc\_means[c] \leftarrow \text{mean}(nc\_vals[c])$
10: **end for**
11: **// Apply the spacing function to determine initial thresholds**
12: $(non\_class\_values, class\_values) \leftarrow \text{spacing}(c\_means, nc\_means)$
13: **// Apply nudge adjustments for better separation**
14: **for** each class c **do**
15:     **if** $non\_class\_values[c] > class\_values[c]$ **then**
16:         $non\_class\_values[c] \leftarrow non\_class\_values[c] + \epsilon$
17:     **else**
18:         $non\_class\_values[c] \leftarrow non\_class\_values[c] - \epsilon$
19:     **end if**
20:     **// Enforce bounds [0,1] with compensatory adjustments**
21:     **if** $non\_class\_values[c] \leq 0$ **then**
22:         $non\_class\_values[c] \leftarrow 0$
23:         $class\_values[c] \leftarrow \min(\epsilon, 1)$
24:     **else if** $non\_class\_values[c] \geq 1$ **then**
25:         $non\_class\_values[c] \leftarrow 1$
26:         $class\_values[c] \leftarrow \max(1 - \epsilon, 0)$
27:     **end if**
28: **end for**
29: **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 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 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 network's 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 soft max-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

1: **// Compute variability measure**
2: $\sigma_{sum} \leftarrow$ sum of standard deviations across all current outputs
3: **for** each class c **do**
4:    **// Check separation requirement**
5:    **if** $|class\_values[c] - non\_class\_values[c]| < \sigma_{sum}$ **then**
6:       **// Adjust non-class value based on relative positioning**
7:       **if** $non\_class\_values[c] \geq class\_values[c]$ **then**
8:          $non\_class\_values[c] \leftarrow non\_class\_values[c] + \lambda\sigma_{sum}$
9:       **else**
10:         $non\_class\_values[c] \leftarrow non\_class\_values[c] - \lambda\sigma_{sum}$
11:       **end if**
12:       **// Handle boundary violations with compensation**
13:       **if** $non\_class\_values[c] > 1$ **then**
14:          $excess \leftarrow non\_class\_values[c] - 1$
15:          $non\_class\_values[c] \leftarrow 1$
16:          $class\_values[c] \leftarrow \max(0, class\_values[c] - excess)$
17:       **else if** $non\_class\_values[c] < 0$ **then**
18:          $deficit \leftarrow |non\_class\_values[c]|$
19:          $non\_class\_values[c] \leftarrow 0$
20:          $class\_values[c] \leftarrow \min(1, class\_values[c] + deficit)$
21:       **end if**
22:    **end if**
23:    **// Final bounds enforcement**
24:    $class\_values[c] \leftarrow \text{clip}(class\_values[c], 0, 1)$
25:    $non\_class\_values[c] \leftarrow \text{clip}(non\_class\_values[c], 0, 1)$
26: **end for**
27: **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.
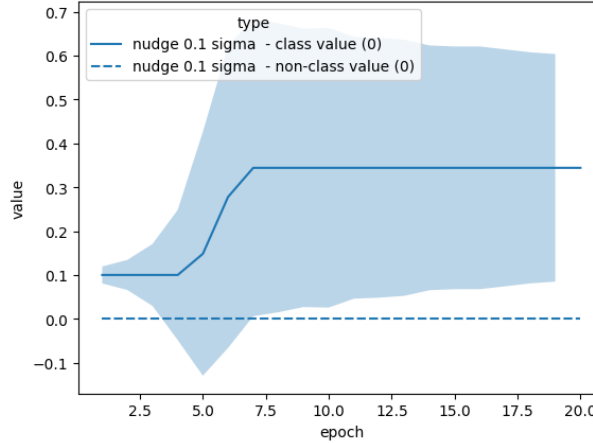
Figure 1: Class and non-class value development over 20 epochs

In Figure 1 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 σ_{sum}, or rather, the area within a distance of σ_{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.

We also observe that, as described in Algorithm 2, when the non-class value comes within a distance of σ_{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 σ_{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.
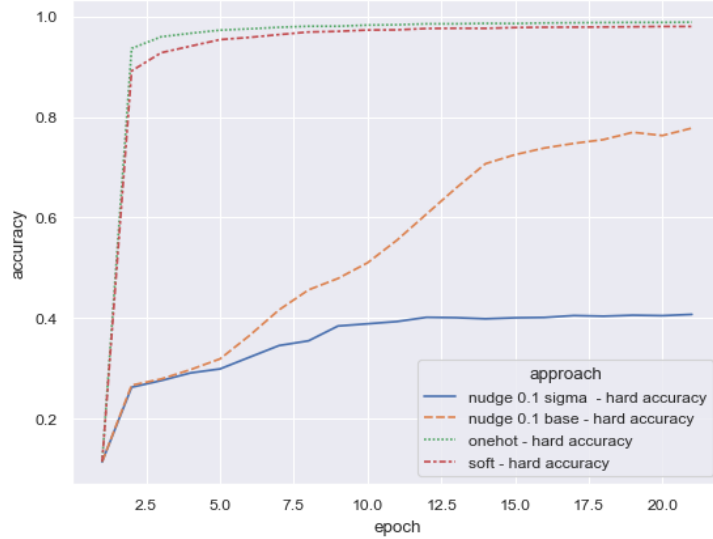
Figure 2: Accuracy Over Time

As shown in Figure 2, 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 3 we compare the development of accuracy with the average distance between class and non-class values in each approach.
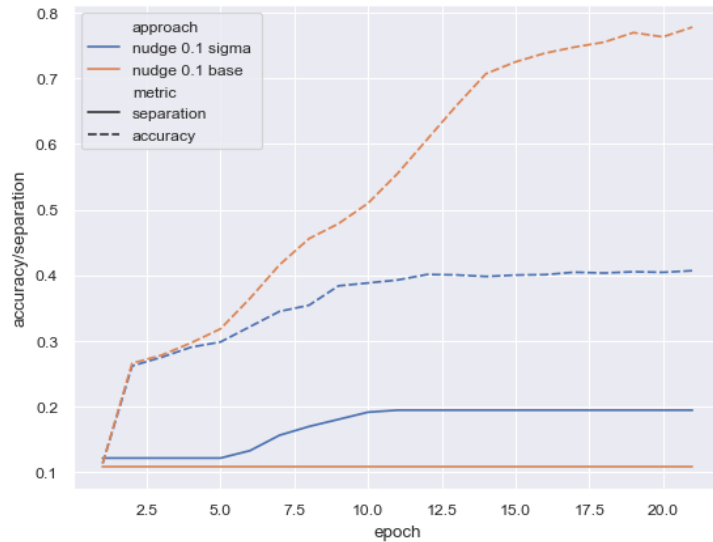

Figure 3: Accuracy vs. Target Value Separation

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 4 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.
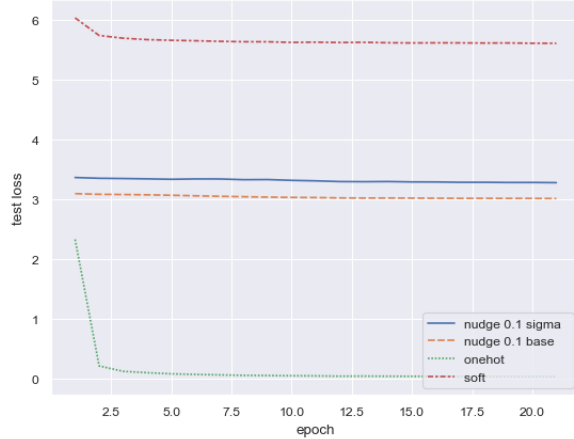
(a) Loss development over training epochs
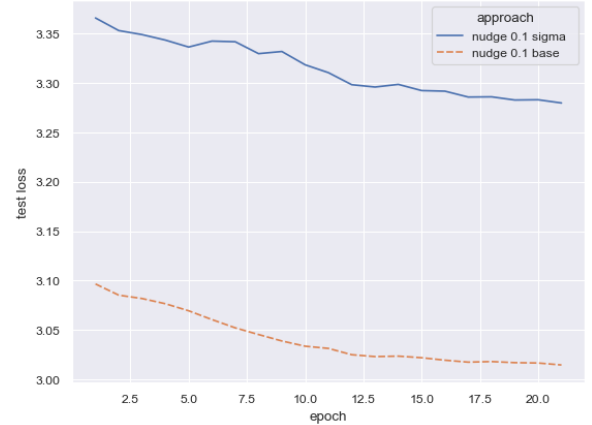
(b) Loss development of base and sigma approaches

Figure 4: 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 5, we compare the final accuracy and confidence of the four different approaches after training for 20 epochs.
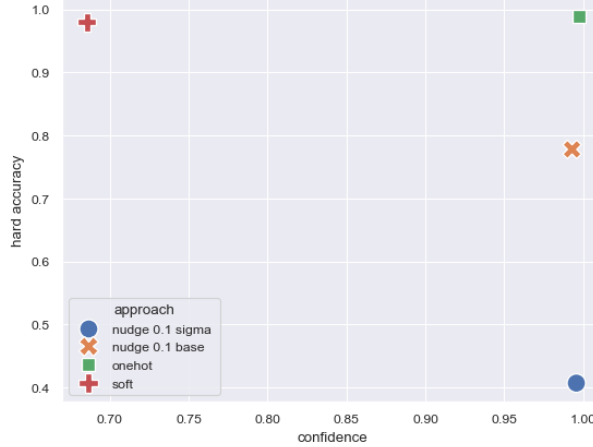
Figure 5: Accuracy vs. confidence

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 5). 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 4(b) clearly shows the loss of our approaches decrease steadily during the entire training process, the accuracy (as seen in Figure 2), 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 3). 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 baseline median-nudge initialized approach without spacing exhibits a more salvageable accuracy / confidence ratio, potentially indicating some merit in the approach if it is developed further.

Similar techniques exist,

6.4   Implications for Future Work

7   Conclusion

8   Links

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