Summary - “An Analysis of Sigma Adaptation of Target Values in ANN Training”

1.) Abstract + Introduction →

In this study, the students explored a new training method for artificial neural networks that focuses on adjusting target values dynamically. Unlike traditional one-hot-encoding methods that use fixed targets like 1 and 0, this sigma adaptation method modifies targets based on the network’s output to improve training speed and accuracy.

Artificial neural networks are a key part of modern machine learning, designed to work like the human brain. They consist of connected neurons in layers, where each neuron assigns weights to inputs and uses an activation function to produce an output. ANNs are especially good at tasks such as image recognition, natural language processing, and classification, as they can learn complex patterns from large datasets.

To train ANNs, input-output pairs from datasets are used. The aim is to reduce the network error, which compares desired outputs (target values) with the actual outputs. The target value indicates the correct output for each input, guiding the network in learning what outputs should look like. Typically, the target values are set to 1 for the correct class and 0 for others, following the winner-takes-all method, where the highest value neuron is activated, representing the correct class.

Training involves several cycles called epochs, during which the entire training set is processed. After each epoch, the neuron closest to the target value is activated. Once training is complete, the network is tested on new data to check its accuracy in classifying unseen patterns.

Traditional training uses fixed target values but can be inefficient, especially with activation functions like the sigmoid function, which can slow down learning due to flat gradients at extremes. To address this, a new method called sigma adaptation dynamically adjusts target values during training. This method adjusts based on the network’s performance to utilize the activation functions more effectively, potentially leading to faster convergence.

The goal of this study is to assess the sigma adaptation method against traditional methods, to see if it can improve training efficiency and accuracy in classification tasks. The upcoming chapters will cover the theoretical aspects of ANNs, explain the sigma adaptation method, and present experimental results.

2.) Artificial Neural Networks →

Artificial Neural Networks are machine learning models that mimic how the brain processes information. They have layers of interconnected neurons that change input data into respective output signals. Each neuron gets inputs, weights, and a bias, then uses an activation function to create an output.

a.) Structure of ANN's

An ANN has three main types of layers:

- Input Layer: This layer has neurons that take in the initial data, with each neuron representing a feature of the input data.
- Hidden Layers: Located between the input and output layers, these layers perform most computations and change inputs into outputs for the next layer. The number of hidden layers and neurons in these layers vary and affect the model performance crucially.
- Output Layer: This layer gives the final prediction, with each neuron representing a possible class in classification tasks. The model adjusts weights during training to maximize the correct class output.

All neurons in a layer connect to every neuron in the previous and next layer, creating a dense network. These connections have weights that are learned during training.

b.) Neurons and Weights

The operation of a single neuron consists of three main steps.

- First, the **Aggregation Function** takes one or more input signals with weights and a bias for each neuron to calculate the weighted sum

$$z_j = \sum_i w_{ij} x_i + b_j$$

- Second, the **Activation Function** takes that sum and applies a nonlinearity, such as the sigmoid function (defined in picture). This allows the network to learn and represent complex patterns.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Lastly, the **Neuron Output** is determined by the activation function applied to the weighted sum of the inputs, e.g. for using the sigmoid function resulting in

$$a_j = \sigma(z_j)$$

where a_j is the output of neuron j.

c.) Sigmoid Activation Function

The sigmoid activation function is widely used in neural networks, especially for binary classification tasks. It maps input values to an output range from 0 to 1, which is helpful for probability representation. The formula for the sigmoid function is

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where e is the base of the natural logarithm. Its output makes it ideal for probability interpretation, with the steepest gradient around 0. 5. However, it has a vanishing gradient problem, where the gradient is very small near 0 or 1, causing minimal weight updates in backpropagation. Small changes in the input near 0.5 result in larger changes in the output, while inputs near the extremes cause very small changes in output. Additionally, its non-zero-centered outputs can lead to inefficient gradient descent updates and slower learning.

d.) Classification

In neural networks, classification assigns input data to specific predefined classes based on their features.
The classification process in ANNs includes several key steps to accurately categorize input data.

- **Data input:** involves initially feeding the input data into the network through the input layer, where each neuron corresponds to a specific feature or attribute. The input layer simply sends raw data to the next layer without performing computations.
- **Feature extraction:** occurs in the hidden layers, where they begin processing the input to extract relevant features. Each neuron computes weighted sums of inputs, adds bias (parameters learned during training), and applies an activation function, often a sigmoid function, to capture complex patterns.
- **Prediction**: processed data goes to the output layer, with neurons representing potential classes. The final prediction is made by selecting the class with the highest output value.

e.) Targets

In neural networks, target values represent the correct output for an input during training. These values are important because they guide the network in learning and adjusting its internal settings, like weights and biases, to reduce the gap between its predicted output and the target. In tasks like classification, target values show the right class for each input. Typically, target values are binary: the correct class gets a value of 1, while others get 0. This binary system forms the winner-takes-all method, where the neuron with the highest output is seen as the winner for the predicted class.

Target values are essential for the network's learning process. By comparing the network's output with these values, the network can change its weights during backpropagation to lower the error, measured by a loss function, such as cross-entropy. Training aims to minimize this error over time, enhancing the network’s ability to predict the right class for new inputs.

Traditionally, target values stay fixed, but this can limit performance with certain activation functions like the sigmoid function (reasons mentioned above). Dynamic target value strategies, like sigma-adaptation, adjust these values during training to improve learning efficiency. This allows the network to refine its predictions as target values change, rather than solely aiming for extreme outputs.

i.) Common Practice in Target Value Assignment

In traditional neural networks for classification, fixed target values are used for output neurons. The correct class is given a target value of 1, while other classes receive 0, creating a 'winner-takes-all' approach. This method helps the network learn to differentiate the correct class during training from incorrect ones. The network adjusts its weights and biases to reduce the difference between predicted outputs and fixed target values, measured by a loss function like *mean squared error* or *cross-entropy loss*. The goal is to minimize this loss, showing the network's predictions are aligning with the targets.

Using fixed target values has important characteristics and implications:

- **Simplicity and Consistency:** They provide a straightforward way to train the network, making it easier to understand and implement. Also, the network can easily interpret which class it should aim to predict.
- **Clear Error Signa**l: The difference between predicted and target values creates a clear error signal, guiding the network's learning through backpropagation, which is crucial for effective learning.
- **Binary Decision Making**: They promote a binary decision-making process, where the network activates the correct class output while deactivating others, making classification straightforward. This helps the network focus on clearly distinguishing between different classes (by pushing outputs towards these extreme values).

However, this method might lead to inefficient learning, requiring more iterations to adjust weights, which can extend training times, especially on complex datasets. Despite this, simplicity and effectiveness make it a common practice, with ongoing efforts to optimize training and improve neural network performance, including adapting target values continuously.

f.) Continuous Adaptation of Target Values

The sigma adaptation method offers a flexible way to adjust target values during training. Instead of using set values of 0 and 1, it changes the targets based on the standard deviation between class and non-class values. The idea behind this is to address the inefficiencies associated with fixed target values. This continuous adaptation aims to improve training efficiency by allowing the network to adjust its outputs as it learns.

i.) Sigma-Adaptation Method

The sigma adaptation method provides a flexible way to set target values for output neurons during training. Instead of using fixed values of 0 and 1, it adjusts the target values based on the network's output in each training epoch. This helps reduce network error from the beginning since the target values are aligned with the network's current output. Setting target values closer to the network's current output makes learning more efficient, reducing the need for drastic early adjustments. The motivation for sigma adaptation is to overcome the limitations of fixed target values in traditional ANN training. Fixed values at the sigmoid function's extremes (0 and 1) are harder to learn. Dynamic target values allow the network to focus on easier values and improve performance. Sigma adaptation does not specifically aim to avoid flat gradient spots; avoiding them may happen as a beneficial side effect. The sigma adaptation method involves the following steps:

- **Initial Target Values**: Start with initial target values that are not at the extremes, so near 0. 5.
- **Dynamic Adjustment**: Adjust the target values after each training epoch based on the network's output. Adjustment is calculated by using the standard deviation (σ) of the network's output values.
- **Gradient Exploitation**: Focus on target values where the sigmoid function's gradient is highest to get efficient learning.
- **Refinement**: Gradually refine the target values towards desired classification outcomes.

These steps aim to improve the learning efficiency of artificial neural networks for classification tasks.

ii.) Sigma Adaptation Process

The Process consists of six steps:

- Calculate the average of class and non-class values →

$$\mu_{\text{class}} = \frac{1}{N_{\text{class}}} \sum_{i=1}^{N_{\text{class}}} x_i$$

$$\mu_{\text{non-class}} = \frac{1}{N_{\text{non-class}}} \sum_{i=1}^{N_{\text{non-class}}} x_i$$

where x_i are the output values, and N_{class} and $N_{\text{non-class}}$ represent the number of class and non-class values, respectively.

- Calculate the Standard Deviations of class and non-class values →

$$\sigma_{\text{class}} = \sqrt{\frac{1}{N_{\text{class}}} \sum_{i=1}^{N_{\text{class}}} (x_i - \mu_{\text{class}})^2}$$

$$\sigma_{\text{non-class}} = \sqrt{\frac{1}{N_{\text{non-class}}} \sum_{i=1}^{N_{\text{non-class}}} (x_i - \mu_{\text{non-class}})^2}$$

- Compute the Sigma Sum →

$$\sigma_{\text{sum}} = \sigma_{\text{class}} + \sigma_{\text{non-class}}$$

- Adjust the Averages →

$$\mu_{\text{class-adjusted}} = \mu_{\text{class}} + \frac{\sigma_{\text{sum}}}{2}$$

$$\mu_{\text{non-class-adjusted}} = \mu_{\text{non-class}} - \frac{\sigma_{\text{sum}}}{2}$$

- Boundary Checks and Adjustments →

If $\mu_{\text{class-adjusted}} > 1$:

$$\mu_{\text{class-adjusted}} = 1$$

$$\mu_{\text{non-class-adjusted}} = \mu_{\text{non-class}} - \frac{\sigma_{\text{sum}}}{2}$$

If $\mu_{\text{non-class-adjusted}} < 0$:

$$\mu_{\text{non-class-adjusted}} = 0$$

$$\mu_{\text{class-adjusted}} = \mu_{\text{class}} + \frac{\sigma_{\text{sum}}}{2}$$

- Check if Further Adjustment is Needed →

"If the existing distance between the averages is already larger than the minimal required distance (i.e., $\mu_{\text{class-adjusted}} - \mu_{\text{non-class-adjusted}} \geq$ minimal distance), no further adjustments are made."

iii.) Alternative 1: Adjust Only Non-Class Values

We only adjust the non-class values, keeping class values stable to ensure reliable classification while allowing flexibility through the non-class values.

$$\mu_{\text{non-class-adjusted}} = \mu_{\text{non-class}} - \frac{\sigma_{\text{sum}}}{2}$$

$$\mu_{\text{class-adjusted}} = \mu_{\text{class}}$$

This helps maintain stable class output while encouraging non-class differentiation.

iv.) Alternative 2: Use Full Sigma Sum Instead of Half

Instead of dividing the sigma sum by two, we can use the full sigma sum in the adjustment process. This change may cause larger shifts in the averages, resulting in more aggressive adjustments. The aim is to possibly speed up convergence by increasing the separation between class and non-class values.

$$\mu_{\text{class-adjusted}} = \mu_{\text{class}} + \sigma_{\text{sum}}$$

$$\mu_{\text{non-class-adjusted}} = \mu_{\text{non-class}} - \sigma_{\text{sum}}$$

v.) Alternative 3: Adjust Only Non-Class Values Using Full Sigma Sum

In this variation, we blend the first two options by changing only the non-class values with the full sigma sum. The goal is to keep the class values fixed while significantly adjusting the non-class values. This approach is helpful when it's important to maintain the accuracy of class predictions while optimizing the separation for non-class predictions.

$$\mu_{\text{non-class-adjusted}} = \mu_{\text{non-class}} - \sigma_{\text{sum}}$$

$$\mu_{\text{class-adjusted}} = \mu_{\text{class}}$$

3.) Experimental Setup →

a.) Parameters

- **Dataset**: In the study, the MNIST dataset is used, which is well-known for image recognition in machine learning. MNIST has hand-written digits from 0 to 9, shown in 28x28 pixel grayscale images. The aim is to correctly recognize the digit in each image. Half of the images come from American Census Bureau employees, and the other half comes from American high school students, making up 60,000 training images and 10,000 testing images for learning and evaluating digit classification models.

- **Model architecture:** The model includes an input layer with 256 neurons (each corresponding to a pixel in the picture), a hidden layer with 32 neurons, and an output layer with 10 neurons for each digit. Each layer is fully connected. Rprop (Resilient Propagation) is used as the training algorithm, adjusting weights based on the gradient's sign for faster convergence.

- **Training:** The training parameters feature a rate of 0. 1, a mini-batch size of 1000, and the cross-entropy loss function, with 70 epochs for the complete dataset. The parameters help determine the speed and stability of the learning process.The model was trained 100 times to ensure robustness and consistency in our results.

- **Evaluation Metrics**: Performance was measured using both test accuracy and network error on the MNIST test set.

- **Hardware & Software**: The hardware included a Nvidia GeForce GTX 1650 Ti and AMD Ryzen 7 4800H, complemented by the Boone Java Framework.

- The team chose these configurations after consulting with the project supervisor, refining through experiments to enhance training speed while maintaining test accuracy.

b.) Evaluation Metrics

The test accuracy of a machine learning model is an important measure of how well it can work with new data. In image classification, it checks how correctly the model predicts labels for a set of 10,000 images. High test accuracy means the model reliably labels images, showing strong performance in real situations.

Network error is also key during training. It measures the difference between the model's predictions and actual labels, often using Cross-Entropy Loss. Lower network error usually suggests better alignment with true labels, but it doesn't always match up with test accuracy. A model can have high network error yet still perform well based on data complexity and overfitting. In this context, it's used as a control mechanism during training (consistently decreasing error → the model is learning effectively, increasing error → potential issues in the algorithm).

Ultimately, test accuracy is the main indicator of the model's performance (reflects capability to classify new data correctly). While network error can provide useful insights during training, especially in identifying potential issues with learning

efficiency, achieving high accuracy on the test set is the main goal, as it reflects the model's real-world predictive ability.

c.) Experimental Results

" Conventional training uses 0 or 1 as class values and is being analyzed in relation to base sigma adaptation and alternatives 1 through 3, with a focus on their effects on accuracy and network error. " → Results in original paper, does not make a lot of sense to summarize them !!

4.) Implications + Summary →

The results from both common practice and adaptive methods show only small differences, with each having benefits based on training goals. Adaptive methods typically have lower error rates, but the accuracy differences are minor, suggesting that these could be random effects rather than real improvements in model performance. The common practice method has moderate accuracy but higher error rates, indicating it struggles more to match predictions with the actual outcomes. This could be due to its reliance on fixed targets, making it slower to adjust during training.
Alternative 2 demonstrates better error reduction while keeping competitive accuracy, yet the error curves for all adaptive methods are quite similar, even when sigma adaptation was expected to show lower errors initially. This might point to issues in implementation or assumptions, such as the minimum distance constraint.
Alternative 3 has the lowest error rates but does not significantly outperform common practice or other adaptive methods regarding accuracy. This shows that reducing error does not always lead to better classification performance. We can see that error minimization does not always correlate with improved accuracy.
Adaptive methods using ½ σ adjustments add variability in accuracy but maintain lower error rates, suggesting a cautious approach may be beneficial.
Overall, while adaptive methods reduce errors compared to common practice, the accuracy differences are small and could be random. Future studies might analyze these differences more thoroughly.

This work examines how sigma adaptation, which changes target values during the training of artificial neural networks, affects training effectiveness and efficiency. Instead of using fixed targets like 0 and 1, sigma adaptation adjusts targets based on the network's outputs and their variability. The goal was to see if this adjustment could improve convergence speed and classification accuracy.
Various sigma-adaptation methods were tested, including different strategies for adjusting target values. While these methods generally led to lower network error rates, they did not consistently result in significantly higher accuracy. The findings suggest that the relationship between network error and accuracy is complicated, and reduced error does not ensure better performance. Ultimately, the changes from sigma adaptation did not provide substantial benefits over fixed target values.