Optimization of Target Values in an Artificial Neural Network

Definitions

- **Class Value** – Target value for the desired class.
- **Non-Class Value** – Target value for the non-desired class.
- **Epochs** – Number of times the entire training dataset is passed through the neural network.
- **Accuracy** – Ratio of correctly classified patterns to the total number of patterns.

What are target values?

In a regular NN used for classification we have some set of input neurons which connect to some number of fully connected layers and eventually end in a set of output neurons. When training the network, we define each one of these output neurons as representing one class. If we are e.g. classifying images into the categories of cat, dog or rat the input neurons would somehow correspond to the pixels of the image while the three output neurons describe what class the data belongs in. We may, as such, input an image of a cat for training with the expectation of getting the output (1,0,0) so 1 in the cat class 0 in all others. As I understand it (1,0,0) are the *target values,* the actual output the network gives for such a cat image is then compared to the expected output (calculating the error between the target values and the values we got) after which this information is used to adjust the network in a way that increases its accuracy.

What is there to optimize?

Just because (1,0,0) for cat, (0,1,0) for dog etc. is easy for us to read and understand does not mean that these are the most effective target values for training a neural network. In my current understanding the target values refer to the actual components of the expected output vector for each class, so as mentioned the target values for cat make up the expected vector of (1, 0, 0). In this example the *class value* is 1 and the *non-class value* is 0. We are not optimizing the target values as such but the class- and non-class values used to construct these target values.

How does this optimization occur?

The paper describes three broad approaches for performing this optimization, the third of these is elaborated on further with three adaptations of the concept.

The first approach is simply picking 1 and 0 for class- and non-class values respectively, while **the second approach** involves manually analyzing the activation function and choosing values that provide better gradient flow (like 0.8 and 0.2 for a sigmoid function.)

The third approach is the main concept outlined in the paper, here we continue to adapt the class- and non-class values during training. In the base case (not using any of the three described variations) it is unclear by what criteria the initial class value is chosen, but the paper does state this:

Before training we run all our samples through the network and choose a class value by some (yet undefined) criterion, this might have something to do with the gradient or simply picking the highest class-value. The non-class value is then made by averaging all the other values we did not choose in the earlier process.

The paper doesn't explicitly state this, but it would be reasonable to assume that we designate one output neuron for each of our classes in this process, then get the class-/non-class values from the respective neurons depending on the current sample's class.

After this pre-training epoch we now have both a guess for the class value as well as the non-class value. We now begin training the network using these values, but each epoch we adjust them as follows:

For each epoch we choose whichever *class value* is closest to the *class value guess* from the previous epoch and store *their average* as **this** epochs class value. The *new* non-class value guess is calculated by taking the average of all of this epoch's non-class values as well as the *previous* epoch's non-class value guess and once again compute *their average*.

Like this we refine the class-value by, as said, always averaging it with the closest value of the current epoch and refine the non-class value by taking the average non-class value across **all epochs.**

Adaptations

The method described above has three adaptations which impose further limitations or change how the non-class value is calculated.

dmax-Adaptation

This adaptation throws out the previously described method of generating the non-class value and instead finds them by comparing the class values to 0.5. If the class value for that epoch lies above 0.5 the non-class value is chosen to be 0, but if the class value is below 0.5 the non-class value is 1. Additionally, this method also constrains the class value to the range of [0, 1].

dmax is defined as lying in this range as well but the paper doesn't explain what the parameter is used for beyond stating that it "allows us to control the separation between class and non-class values, while the decision rules and bounds maintain the consistency and stability of the training process."

ϵmin-Adaptation

This adaptation, unlike dmax, doesn't directly replace the usual method for calculating the non-class values, instead it looks at the distance between the class and non-class values and then adjusts the non-class value if they get too close. The minimum distance between the values and the amount the non-class values are adjusted by are defined by the same parameter ϵmin. We can choose any number [0,1] for it. When the distance between class and non-class values falls short of this ϵmin we move the non-class value either up by ϵmin (so non-class + ϵmin) or down by ϵmin (non-class - ϵmin) which we choose depends on whether non-class =/- ϵmin is less than 0 or greater than 1, here we use whichever option keeps us within the range of [0,1]. What happens when we choose an ϵmin such that neither condition is met but the distance between class and non-class values is too small? The paper does not elaborate on this, but it would probably be reasonable to choose whichever operation increases the distance between the class and non-class values the most in this case. Additionally, this method also ensures that the class and non-class values stay within [0,1].

σ-Adaptation

In this adaptation we calculate our class and non-class value as normal but then also get the sum of the standard deviations for that epoch, we call this value σ-sum. We now make sure that 1. The class and non-class values remain within [0,1].

2. The class and non-class values are always at least σ-sum/2 apart.

In this process the paper states that we continuously move non-class values down by σ-sum/2 while class values are only moved up, also by σ-sum/2. The exception to this is when we need to prevent one of the values from overshooting the bounds, in that case only one of the values is moved or, if necessary, the value which would otherwise fall outside the bounds is forcibly set to 0 or 1.

It is important to note that all these adjustments only take place when σ-sum is greater than the distance between the current class and non-class values.

Test Results

Metric	Worst				Best
Accuracy	ϵmin	dmax	σ	Custom constant values	1 and 0

Overall, some of the methods provided decent performance but with a significantly reduced number of epochs to reach that level of quality, most of the methods seem to perform unacceptably poorly however, possibly due to the amount of fluctuation in the non-class values. It is also noteworthy that taking the nature of the activation function into account and picking values better suited to it (as compared to the default 0 and 1) allows for slightly better training accuracy and more confidence in the predictions but doesn't appear to positively affect the test accuracy, at least in this experiment.

Resources?

- Paper on BB**: adaptiveTargets**