Problems and Solutions of Visualizing and Analyzing Multidimensional Output from MLP Networks - Barycentric Projections.

Filip Piękniewski and Leszek Rybicki

Departament of Mathematics and Computer Science, Nicholaus Copernicus University,
Toruń, Poland, http://www.mat.uni.torun.pl

Abstract. Barycentric plotting, achieved by placing gaussian kernels in distant corners of the feature space and projecting multidimensional output of neural network on a plane, provides information about the process of training and certain features of the network. Additional visual guides added to the plot show tendencies and irregularities in the training process.

1 Introduction

One of the important problems of modern neural network theory is to find a good measure, or a method for determining if the network is well trained and ready to solve "real life" problems, or if there are singularities that could render the network useless. The backprop algorithm (and other learning algorithms) minimizes a well defined error function, usually the Mean Square Error (MSE). One might ask, since the the value of the error function is so well defined and easy to calculate, what more can one need? In fact, mean square error (and other error functions) provides just a piece of statistical information about the learning process, while many important issues are lost. Neither the MSE nor any other real-valued measure is sufficient to tackle all relevant issues.

To get more information about the learning process, it would be necessary to analyze the whole output data, find out which vectors are problematic, which of them separate well and which don't. Obviously, analyzing thousands of numbers by hand is not a good idea. Economy, biology and physics know a common solution to that - plotting.

2 Visualization

From now on, we will focus on MLP networks, used for classification. The problem can be defined as follows:

- Input consists of n vectors $E^{(i)} \in \mathbb{R}^s$, each of them is assigned to one of k categories. Vector $E^{(i)}$ is assigned to category $Cat(i)$
- Network consists of two layers. There are s inputs, some number h of hidden neurons, and k output neurons.

- If the input vector $E^{(i)}$ is assigned to category t, then the network is trained to activate the t-th output neuron, while others should not be activated. The desired output vector corresponding to the i-th category will be denoted by $\overrightarrow{Cat(i)}$ as opposed to the actual network output $O^{(i)}$.

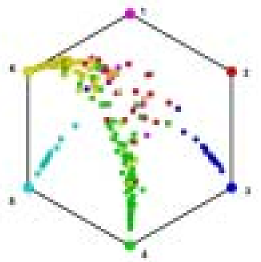

Fig. 1. A plot with six categories and 214 patterns of data. It visualizes the output of a sigmoidal MLP network with a MSE of 0.214. The plot was scaled by $\sigma = 0.35$.

For example, if there are two categories then the output is two dimensional, and the categories are mapped onto vertices $(1, 0)$ and $(0, 1)$ of the unit square, which coincides in this case with the whole activation space. The activation space in general is a k-dimensional unit hypercube I_k, with $\overrightarrow{Cat(i)}$ concentrated on the bisecting hyperplane H_k of the main diagonal $[\overrightarrow{0}, \overrightarrow{1}]$ of I_k.

First of all, let's review the problem, pointing out important issues:

- We have n vector outputs in I_k which lay inside a k-dimensional hypercube.
- We map the categories onto corresponding vertices.
- The training process should result in most of the data clustering around these vertices.

Therefore the most effective method of visualization seems to be some kind of a projection. Now, what interests us the most about each vector $O^{(i)}$ is:

- Is it well classified (in other words, is the distance between $O^{(i)}$ and $\overrightarrow{Cat(i)}$ minimized among the categories)?
- Is it far from or close to its assigned category; does it tend to move towards another category?
- Is it an outlier or are there any other vectors in its vicinity?

Let's consider a mapping system, as follows. Each of the categories is a center of a Gaussian radial function

$$G(x; a, \sigma) = e^{-\frac{(x-a)^2}{2\sigma^2}}$$

which will be used for scaling. Categories will be mapped, one-to-one, onto the corners of a polygon. Assume there are k categories. Inside a k-gon, each output vector $O^{(i)}$ is projected as follows:

$$\begin{aligned} O_x^{(i)} &= \frac{1}{\delta} \sum_{l=1}^{k} G\left(\left(\left\| O^{(i)} - \overrightarrow{Cat(l)} \right\|\right); 0, \sigma\right) \cdot \overrightarrow{Cat(l)}_x \\ O_y^{(i)} &= \frac{1}{\delta} \sum_{l=1}^{k} G\left(\left(\left\| O^{(i)} - \overrightarrow{Cat(l)} \right\|\right); 0, \sigma\right) \cdot \overrightarrow{Cat(l)}_y, \end{aligned} \tag{1}$$

where $\delta = \sum_{l=1}^{k} G\left(\left(\left\| O^{(i)} - \overrightarrow{Cat(l)} \right\|\right); 0, \sigma\right)$ is a normalizing factor, $(O_x^{(i)}, O_y^{(i)})$ are coordinates of the i-th output's projection, $(\overrightarrow{Cat(l)}_x, \overrightarrow{Cat(l)}_y)$ are coordinates of the l-th category projection (l-th vertex of k-gon), $\|\|$ is the Euclidean norm in a k-dimensional space.
A sample plot can be seen in Fig. 1. To make the plot more useful, some extra information coded as color has been added, defining where the sample belongs, and how it was classified. Further in the paper there will be some other add-ons like network dynamics, and convex hull around each data cluster.

Since the plot mechanism described above is just a projection, some information is lost. Two dots displayed as close to each other can be quite distant in the activation space. If dots from different categories tend to mix up in the plot, it doesn't always mean they mix in the activation space. These are obvious faults of plotting multidimensional data into two dimensions. One way to compensate for these faults is to look at different plots, and choose the one that gives the best view. This can be done by permuting the vertices of the polygon (which correspond to category centers). Unfortunately it's rather difficult to numerically choose an optimal (most suitable for a human) permutation, so the problem of choosing the best permutation is left to the plot reviewer (user).

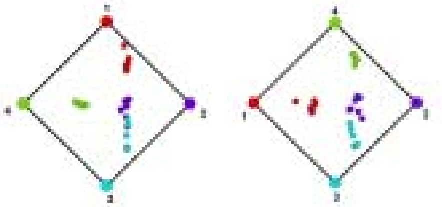

Fig. 2. Two plots, showing exactly the same data (sigmoidal MPL output, MSE 0.1, scaled by a factor $\sigma = 0.7$), but with different permutations of category vertices. Please notice that category 2 and 3 don't mix.

Permuting categories is just one of many methods to make such plots more useful. We introduced some other self-adapting optimizations, that might make the plot just a bit better. The parameter σ, responsible for dispersion, is constant for all categories and data samples. What if σ is made dependent upon some parameters specific for a category? Put:

$$\sigma_{(l)} = \sigma_0 \max_{i \in \mathbb{N}, Cat(i) = l} \left\| O^{(i)} - \overrightarrow{Cat(l)} \right\| \tag{2}$$

In this case, the dispersion of the Gaussian kernel depends on the maximal distance between the center of a category l, and vectors assigned to l. If the

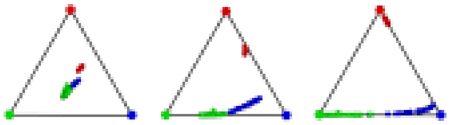

Fig. 3. Three plots created using the same data (iris data parsed by a MLP network, MSE=0.086) but with different scaling options. The one on the left is scaled with $\sigma = 2.0$, constant for all categories (1), the one in the middle is scaled using the maximum formula (2) with σ_0 set to 2.0, third one is scaled using the average formula (3) again with $\sigma_0 = 2.0$.

vectors of category l are spread widely all over the activation space, the corresponding Gaussian function will have a big dispersion. A big dispersion means that the category will attract other vectors stronger. What consequences does it have for our plot? Since a ”wide” category is a stronger attractor, this scaling will reveal the border regions in activation space, while most vectors from the l-th category will be projected close to the polygon corners. This kind of projection is good for analyzing border regions of networks that are already trained, and is rather useless for fresh networks.

Another variant of adaptive, category dependent scaling is done by making σ depend on the average distance of vectors from their assigned category. This can be done as follows:

$$\sigma_{(l)} = \sigma_0 \left(\frac{1}{M} \sum_{Cat(i)=l} \left\| O^{(i)} - \overrightarrow{Cat(l)} \right\| \right) \tag{3}$$

As opposed to the previous method, suppose that most vectors are properly classified around their common category, and a few are not. The average distance between sample vectors and the category center is small, and so the σ parameter is small. It’s as if those wrongly classified vectors were treated as data errors rather than network errors. This gives a slightly different projection that displays badly classified samples.

3 Additional visual guides

Adaptive scaling methods and vertex permutations are helpful when it comes to enhancing the informational value of the plot. We suggest several further enhancements.

It’s not uncommon for a single data sample to float away from its category. The network would keep the MSE low by keeping the remaining samples close to their proper categories. While this might be satisfactory from a statistical point of view, the consequences might be catastrophical in real life applications. Imagine a patient with a very uncommon symptoms (due to genetic reasons for example) being categorized to (and treated for) a different disease.

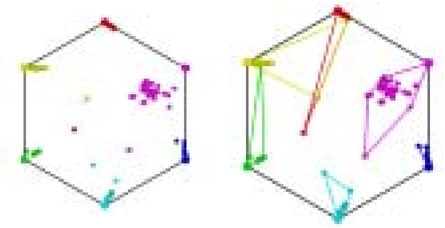

Fig. 4. The 5 categories are well separated... or are they? Convex hulls reval the problem.

An outlier sample could be unnoticed as its corresponding dot could be covered by other dots on the plot. To resolve this, as well as to provide a certain per-category measure of data similarity, the plot can be complemented with convex hulls marking the borders of each category. If a data sample happens to stand out, the hull will expand to contain it, making it clear that the category does mix up with another one.

From a non-overtrained, well generalizing neural net, it is expected that the convex hull of a given category should remain within a vicinity of the category's vertex. If two convex hulls overlap, it suggests that their categories might have been confused by the network, although not necessarily. If the convex hull is stretched away from the category vertex, but the samples remain tucked tight, it means that the net has trouble separating the category and its architecture and/or training process has to be reviewed.

To provide an extra information about the overlapping of categories, simple Voronoi border capability can be added to the plot. The borders separate only the category centers, yet they give a good view of relations between categories. One thing has to be put straight though. The fact that a sample is not in its assigned Voronoi cell, doesn't mean that it is not properly classified. The position of a sample in the plot depends on multiple factors, including relative weights of the output connections and adaptive scaling (σ parameters). Therefore the badly classified samples have been represented by an X symbol. The X mark is coloured with the color of the category it was (mis)assigned to. This gives extra information about misleading data or patterns misassigned in the process of data preparation. The X mark is independent of scaling, because its color coincides with the color of the vertex the dot approaches when σ approaches zero.

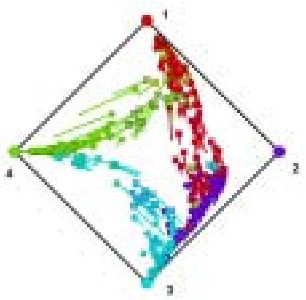

Fig. 5. The dynamics of a network's output in early stages of the training process (backprop algorithm with momentum), MSE=0.3, scaled adaptively (average) $\sigma_0 = 0.5$.

All the enhancements and visual guides described so far are meant to emphasize certain qualities of trained networks or to compensate for the limitations of dimension-reducing projection. To view the dynamic aspects of the training process, the user has either to watch the changing plot during training, or look at multiple plots representing the states of the network in subsequent stages of the process. Displaying multiple plots requires additional space and doesn't provide any visual clue on how far a given dot has moved (or if it has moved at all).

The solution to that is rather simple. The movement of a dot during a certain (user-defined) number of training epochs can be marked as a segment ending at this dot. This might not say much about an individual dot, but the whole plot seems to be enriched by another dimension: time. Extra information given by such a plot is useful in many ways. We know that the training process is going well if the dots are moving towards their assigned categories, but that is a pretty optimistic situation and doesn't really happen throughout most of the process.

The network separates the data well if a group of dots belonging to the same category is moving simultaneously. Any dot described as an outlier before, or a dot that is to become an outlier, would most certainly move in a different direction. In some situations one might observe that all categories move towards one that is rather stable. That means that the network has trouble separating that category and tends to decrease the MSE by warping the whole output in that direction. A situation like that suggests that maybe crossvalidation or data pre-separation might be required. The network might be too small or a committee of networks may be needed to solve the problem.

4 Acknowledgements

We thank our mentor, prof. Włodzisław Duch, who had suggested to take on this project. Great thanks go to dr Norbert Jankowski, who provided us with practical information about the implementation of the visualization module. Dr Tomasz Schreiber gained our gratitude by helping with various theoretical issues and helping us to solve optimalization problems. Last, but not least, we want to thank Paolo Marrone, the author of the JOONE engine.

References

1. W. Duch *Uncertainty of data, fuzzy membership functions, and multi-layer perceptrons (2003, subm. to IEEE Transactions on Neural Networks*
2. W. Duch *Coloring black boxes: visualization of neural network decisions.* Int. Joint Conf. on Neural Networks, Portland, Oregon, 2003, Vol. I, pp. 1735-1740
3. Paolo Marrone *Java object oriented neural engine* www.joone.org
4. Stanisław Osowski *Sieci neuronowe do przetwarzania informacji*. Politechnika Warszawska, Warszawa 2000
5. J. Korbicz, A. Obuchowicz, D. Uciński *Sztuczne sieci neuronowe, Podstawy i zastosowania*. Akad. Oficyna Wyd. PLJ, Warszawa 1994
6. Robert A. Kosiński *Sztuczne sieci neuronowe, Dynamika nieliniowa i chaos*. Wyd. Nauk-Tech, Warszawa 2002
7. Stanisław Osowski *Sieci neuronowe w ujęciu algorytmicznym*. Wyd. Nauk-Tech, Warszawa 1996
8. W. Duch, J. Korbicz, L. Rutkowski, R. Tadeusiewicz *Biocybernetyka i inżynieria biomedyczna 2000 - tom 6 Sieci neuronowe* Akademicka Oficyna Wydawnicza Exit, Warszawa 2000.