Why Linear and Sigmoid Last Layers Work Better in Classification

Lehel Dénes-Fazakas  
*Obuda University*, denes-fazakas.lehel@uni-obuda.hu

Lásló Szilágyi  
*Sapientia University*, lalo@ms.sapientia.ro

Vladik Kreinovich  
*The University of Texas at El Paso*, vladik@utep.edu

Follow this and additional works at: [https://scholarworks.utep.edu/cs_techrep](https://scholarworks.utep.edu/cs_techrep)

Part of the Computer Sciences Commons, and the Mathematics Commons

Comments:
Technical Report: UTEP-CS-24-08

**Recommended Citation**

[https://scholarworks.utep.edu/cs_techrep/1864](https://scholarworks.utep.edu/cs_techrep/1864)

This Article is brought to you for free and open access by the Computer Science at ScholarWorks@UTEP. It has been accepted for inclusion in Departmental Technical Reports (CS) by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.
Why Linear and Sigmoid Last Layers Work Better in Classification

Lehel Dénes-Fazakas, László Szilágyi, and Vladik Kreinovich

Abstract Usually, when a deep neural network is used to classify objects, its last layer computes the softmax. Our empirical results show we can improve the classification results if instead, we have linear or sigmoid last layer. In this paper, we provide an explanation for this empirical phenomenon.

1 Formulation of the Problem

Classification: a practical problem. In many practical situations, we need to classify an object into one of the given classes – e.g., we need to decide whether a given picture is a picture of a cat, of a dog, or of a bird. In such cases, we have an input $i_1, \ldots, i_N$ consisting of several real numbers – e.g., the intensities of different pixels – and, based on this input, we need to decide to which class the corresponding object belongs.
**How deep learning is used for classification.** In many such situations, we have a large number of objects for which we know the correct class. In such situations, one of the most effective ways to classify is by using deep learning; see, e.g., [1].

In deep learning, data processing is performed by computational units called neurons. Each neuron takes several inputs $a_1, \ldots, a_k$ and transforms them into the value

$$b = s(w_1 \cdot a_1 + \ldots + w_k \cdot a_k + w_0)$$

(1)

for some values $w_i$, where $s(z)$ is a function known as activation function. Deep learning algorithms mostly use the rectified linear activation function (ReLU) $s(z) = \max(0, z)$.

In general, in deep learning, neurons are divided into Layers $1, 2, \ldots, L$:

- neurons from the first layer process the input values $i_1, \ldots, i_N$, and
- neurons form each layer $\ell + 1$ take, as inputs, the outputs of the neurons of the previous layer.

Deep neural networks used for classification are slightly different. Specifically, all the neurons up to the last-but-one layer $L - 1$ use ReLU activation functions. The last-but-one layer of a neural network has as many neurons as there are classes. Let us denote the number of classes by $n$ and the outputs of the corresponding layers by $x_1, \ldots, x_n$. Then, the last $L$-th layer is usually the softmax layer, i.e., the layer that transforms, for some fixed number $a > 0$, the signals $x_1, \ldots, x_n$ into the values

$$p_i = \frac{\exp(a \cdot x_i)}{\sum_{j=1}^{n} \exp(a \cdot x_j)}.$$  

(2)

The values are non-negative and add up to 1. Thus, each value $p_i$ can be naturally interpreted as the probability that the object belongs to the $i$-th class.

In many practical situations, we just want to select the most probable class. In this case, it is reasonable to select the class $i$ for which the probability $p_i$ is the largest.

**It is desirable to do better.** While the usual deep learning algorithm provides a reasonable classification, the resulting classification is not perfect: it sometimes predicts the wrong class. How can we improve the results?

**How can we do better: idea.** Sometimes, the use of a different activation function can lead to better results. This is, for example, why ReLU activation function is now most widely used – it turned out to lead to a better training that the previously used sigmoid activation function

$$s(z) = \frac{1}{1 + \exp(-z)}.$$  

(3)

A specific activation function used in classification is softmax. So, a natural idea is to replace it with some other activation function. One of the reasons for selecting
softmax was that in situations when we assign probabilities to each class, these probabilities should add up to 1 – and softmax guarantees this. However, in frequent situation when we are only interested in selected the most probable class, there is no need for the outputs to add up to 1. So, we can use other well-tested activation functions.

**Our experimental results.** Following the above idea, we replaced the softmax function in the last layer with the sigmoid function, i.e., used the formula

\[ p_i = s(w_1 \cdot x_1 + \ldots + w_n \cdot x_n + w_0), \]  

(4)

where \( s(z) \) is the sigmoid activation function (3). Then, we selected the class \( i \) for which the resulting value \( p_i \) is the largest. Interestingly, we immediately got better results than with softmax.

We got similar improvements when in the formula (4) describing the last layer, we used linear neurons, with \( s(z) = z \).

**Resulting challenge.** How can we explain this empirical result? Without such an explanation, we are not sure whether our results are mostly accidental or whether indeed show that sigmoid and linear last layers work better in classification.

**What we do in this paper.** In this paper, we provide an explanation for our empirical results.

## 2 Our Explanation

**What happens in the traditional approach: analysis.** In the traditional softmax-based approach, we select the class \( i \) for which the value \( p_i \) – as computed by the formula (2) – is the largest possible, i.e., when \( p_i \geq p_j \) for all \( j \).

What does it means in terms of the outputs \( x_i \) of the last-but-one layer? All the values \( p_i \) has the same positive denominator. Thus, \( p_i \geq p_j \) if and only if the numerator of \( p_i \) is larger than or qual to the numerator of \( p_j \), i.e., if and only if

\[ \exp(a \cdot x_i) \geq \exp(a \cdot x_j). \]  

(5)

For \( a > 0 \), the function \( \exp(a \cdot z) \) is strictly increasing. Thus, the inequality (5) is equivalent to \( x_i \geq x_j \).

Thus, in the situations when we are interested only in selecting the most probable class, we do not really need the softmax layer: selecting the class with the largest probability is equivalent to simply selecting the class with the largest possible value of the output \( x_i \) of the last-but-one layer.

**Why this is a problem: case of ReLU neurons.** When the neurons in the last-but-one layer are ReLU neurons, their outputs \( x_i \) are related to the outputs \( z_1, \ldots, z_m \) of the previous layer by a formula

\[ x_i = \max(0, z_i). \]
\[ x_i = \max(0, w_{i1} \cdot z_1 + \ldots + w_{im} \cdot z_m + w_{i0}) \]  

(6)

for some values \( w_{ij} \). In other words, each value \( x_i \) is either equal to 0 or to a linear combination of the values \( z_j \).

When values \( x_i \) and \( x_j \) are both linear functions of \( z_j \), then the equality \( x_i \geq x_j \) takes the form

\[ w_{i1} \cdot z_1 + \ldots + w_{im} \cdot z_m + w_{i0} \geq w_{j1} \cdot z_1 + \ldots + w_{jm} \cdot z_m + w_{j0} . \]  

(7)

In training, we usually start with random weights \( w_{ij} \), and, as a result, trained weights also keep some element of randomness. When the weights are random, with some probability distribution with a continuous density function, then the probability of having an equality between the left- and right-hand sides of the formula (7) is 0. So, with probability 1, for each \( i \) and \( j \), we always have either \( x_i > x_j \) or \( x_i < x_j \).

But what happens if both \( x_i \) and \( x_j \) are zeros? This is not a rare case. Indeed, the value \( \max(0, z) \) is equal to 0 for half of the real line. So we can conclude that in half of the cases, this value is 0. Correspondingly, it is reasonable to conclude that the probability that both \( x_i \) and \( x_j \) are 0s is equal to \( 1/2 \cdot 1/2 = 1/4 \). So, in one quarter of all cases, both values \( x_i \) and \( x_j \) are zeros and thus, we will not be able to decide which class is more probable. And if we select one of these two at random, then in half of such cases, we will make a mistake.

What if we use sigmoid or linear activation function? To answer this question, let us first take into account that the sigmoid function (3) is also strictly increasing. Thus, comparing the values

\[ p_i = s(w_{i1} \cdot x_1 + \ldots + w_{in} \cdot x_n + w_{i0}) \]

and

\[ p_j = s(w_{j1} \cdot x_1 + \ldots + w_{jn} \cdot x_n + w_{j0}) \]

is equivalent to comparing the corresponding linear combinations. In other words, \( p_i \geq p_j \) if and only if

\[ w_{i1} \cdot x_1 + \ldots + w_{in} \cdot x_n + w_{i0} \geq w_{j1} \cdot x_1 + \ldots + w_{jn} \cdot x_n + w_{j0} . \]  

(8)

In other words, in our classification problems, the use of the sigmoid last layer leads to the same result as the use of a linear layer. Because of this, in the following text, we will only consider linear last layer.

For the linear last layer, as we have shown, the inequality \( p_i \geq p_j \) is equivalent to the linear inequality (8). And, as we have argued earlier, for such a linear inequality, with probability 1, we always have either \( p_i > p_j \) or \( p_j > p_i \).

**Summarizing.** In the traditional deep learning approach to classification, when we use softmax in the last layer, there is a positive probability that we will not be able to decide which class is more probable. This probability is equal to 1/4 if we use ReLu neurons in all other layers, and it is even higher if we use the Bump activation function in all other layers.
In contrast, if we use sigmoid or linear last layer, then with probability 1 we can always decide which class is more probable. This explains why in our experiments, the use of sigmoid or linear last layers led to better classification results.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395, and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References