CODING WITH LOGISTIC SOFTMAX SPARSE UNITS

Gustavo A. Lado and Enrique C. Segura

Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires, Argentina

ABSTRACT

This paper presents a new technique for efficient coding of highly dimensional vectors, overcoming the typical drawbacks of classical approaches, both, the type of local representations and those of distributed codifications. The main advantages and disadvantages of these classical approaches are revised and a novel, fully parameterized strategy, is introduced to obtain representations of intermediate levels of locality and sparsity, according to the necessities of the particular problem to deal with. The proposed method, called COLOSSUS (COding with LOgistic Softmax Sparse UnitS) is based on an algorithm that permits a smooth transition between both extreme behaviours -local, distributed- via a parameter that regulates the sparsity of the representation. The activation function is of the logistic type. We propose an appropriate cost function and derive a learning rule which happens to be similar to the Oja's Hebbian learning rule. Experiments are reported showing the efficiency of the proposed technique.

KEYWORDS

Neural Networks, Sparse Coding, Autoencoders

1. INTRODUCTION

One of the properties of simple autoencoders is to produce feature vectors in the hidden layer [1]. The type of response usually produced by these models is known as distributed coding, since activation levels are distributed more or less uniformly over all units [2, 3]. Distributed coding is not the only way to represent feature vectors and, in many cases, it is not the best. Alternatively, there is also the so-called local coding, in which, generally, only one unit is activated at a time [4]. On the other hand, there is the sparse coding which, in some way, works as an intermediate mode between local and distributed coding; since it is characterized by higher activation levels for only a portion of the units at a time, while the activation of the others tends to zero. In many cases sparse coding is considered the best option to represent feature vectors [5]. This paper explores the possibilities of various coding strategies and presents a novel method that has several advantages over other techniques commonly used, providing competitive results.

2. BACKGROUND AND MOTIVATION

Local coding is generally used in classification problems. In this case, the active unit indicates to which category the input belongs, or estimated probabilities are given for the input belonging to each class. For instance, in the activation function known as winner-takes-all [4, 6] a single unit takes the value of 1 while the others remain on 0. Alternatively, the softmax function computes a

David C. Wyld et al. (Eds): CCSEA, BIoT, DKMP, CLOUD, NLCAI, SIPRO, BDML, AIFU - 2020
pp. 245-252, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101019

probability distribution, where the unit corresponding to the class of the input is expected to assume the highest level of activation [7].

On the other hand, there are also several methods to produce sparse coding. A relatively simple method is the so-called Rectified Linear Units (ReLU), where the activation is determined by the rectified function f(x)=max(0, x), used in combination with the inclusion of a penalty term in the cost function, for example L1 norm on the activation values over the output, weighted by a parameter λ to regulate the level of dispersion. This makes training difficult because learning must be done by alternating between these two terms of the new cost function, which also does not guarantee good coding during testing [8, 9, 10].

A more recent variant is the strategy known as top-k sparse coding where, like the ReLUs, the k maximum activation values are preserved and all the others are set to 0. This ensures that there are always exactly k active units. However, the selection of these values makes implementation impractical and generally less efficient. In both cases the use of the max function is because of its computing speed. However, the unbounded nature of this function, in combination with training methods such as back-propagation, can lead to unstable configurations, such as disproportionate weight growth [11].

Finally, distributed coding is perhaps the most common one, since it is the type of representation obtained, for example, in the hidden layers of multi-layer perceptrons [12, 13]. As aforementioned, autoencoders produce this type of codification precisely because they are formed by layers of perceptrons and their response is the result of activation in their hidden layer. Nevertheless, there are also some variants of the simple autoencoder such as the so-called tied, where the weights between the encoder and the decoder are shared, or the denoise, where noise is added only to the input. These alternative techniques produce different types of encodings while using essentially the same architecture.

Each of these strategies shows interesting properties and, at the same time, different limitations, especially when looking for an encoding that meets the specific conditions to capture different types of features. In the following section an alternative strategy is presented to generate the desired type of sparse coding, trying to keep some of the advantages already mentioned, and avoiding the main drawbacks described.

3. REGULATED SPARSE REPRESENTATION

Given that sparse coding can be seen as a compromise between local and distributed coding, we consider the possibility of a sparse coding strategy whose activation ratio can be chosen so that it can behave with any level of sparseness between both extremes.

One type of commonly used activation function that can generate a local coding is softmax. It can be used in statistics to represent a distribution of categories, i.e. the distribution of probabilities over N possible categories. This type of function is also known as multinomial logistic regression, since it is a generalization of the logistic regression (in which case N=2). On the other hand, the logistic function is usually used as activation function to generate distributed encodings, for example in hidden layers in multi-layer perceptrons.

$$\Pr(Y_j = k) = \frac{e^{\beta_k \cdot x_j}}{\sum_h^N e^{\beta_h \cdot x_j}}$$

$$\Pr(Y_j = 0) = \frac{e^{\beta_0 \cdot x_j}}{e^{\beta_0 \cdot x_j} + e^{\beta_1 \cdot x_j}} = \frac{e^{-\beta \cdot x_j}}{1 + e^{-\beta \cdot x_j}} \qquad \beta = -(\beta_0 - \beta_1)$$

$$\Pr(Y_j = 1) = 1 - \Pr(Y_j = 0) = \frac{1}{1 + e^{-\beta \cdot x_j}}$$

This illustrates the relationship between the two extreme types of encoding (distributed and local), even when the type of solution we look for is not exactly a probability distribution. It would be preferable some parameter that permits a smooth transition between both extreme behaviours.

For this, we will consider a network architecture consisting of an input layer X of dimension M and an output layer Y of dimension N, preferably with M>N. Layer X can include an extra unit clamped at 1 that represents an activation threshold. The units in Y are expected to take values on the interval [0,1]. The connections between X and Y are represented by the matrix W, which is initialized at random. The stimulus received by each unit Yj can be expressed as:

The exponential of the dot product it produces a higher stimulus for the units whose weights resemble, at least partially, certain features present at the input. In order to control which portion of the units will have a better chance of being activated, a parameter k will be used. For example, for a value of k=1, a behaviour similar to that of the softmax function is expected, with a single unit taking a higher level of activation than the others. On the other hand, with a value of $k = {}^{N}/_{2}$ it would be similar to the logistic function, with all units having roughly the same possibilities of activation. However, this parameter does not directly indicate how many units will be active at a time, and does not even have to be limited to integer values.

The transfer function is a logistic shifted in ${}^{1}/_{2}$, with a relatively steep β slope, for example with

$$s(x) = \frac{1}{1 + e^{-\beta(x - \frac{1}{2})}}$$

values between 5 and 10, so that it saturates rapidly when approaching 0 or 1.

In order to estimate the level of activation of the units, one must first compute the average stimulus they get.

In this manner it is possible to estimate a maximum activation level (2z) and from there determine a cut-off point p being k steps below this estimated maximum ($k\,.\ {}^{2z}/_{2}$). That is, the

$$\tilde{z}\ \ \tilde{z} = \frac{\sum_h^N z_h}{N}$$

units with a stimulus higher than p will tend to be activated.

$$p = 2\tilde{z} - k\frac{2\tilde{z}}{N}$$

$$= 2\tilde{z}(1 - \frac{k}{N})$$

Finally, for this value to coincide with the inflection point of the transfer function, the stimuli can be multiplied by a correction factor c so that $c \cdot p = 1/2$.

$$c \cdot 2\tilde{z}(1 - \frac{k}{N}) = \frac{1}{2}$$

$$c = \frac{1}{\tilde{z}} \cdot \frac{1}{4 \cdot (1 - k/N)}$$

$$c = \frac{1}{\sum z_h} \cdot \frac{N^2}{4 \cdot (N - k)}$$

Thus the final activation level for unit j is given by:

$$Y_j = s\left(\frac{z_j}{\sum_h^N z_h} \cdot \frac{N^2}{4(N-k)}\right)$$

In the figure 1 the stimulus received by 16 units can be seen along with the estimated value of *p* for *k = 4*, and its effect on the final activation levels. The most active units are those that receive the greater correction, so that the next time a similar pattern appears in the input, the differences in the activation levels will be even more emphasized.

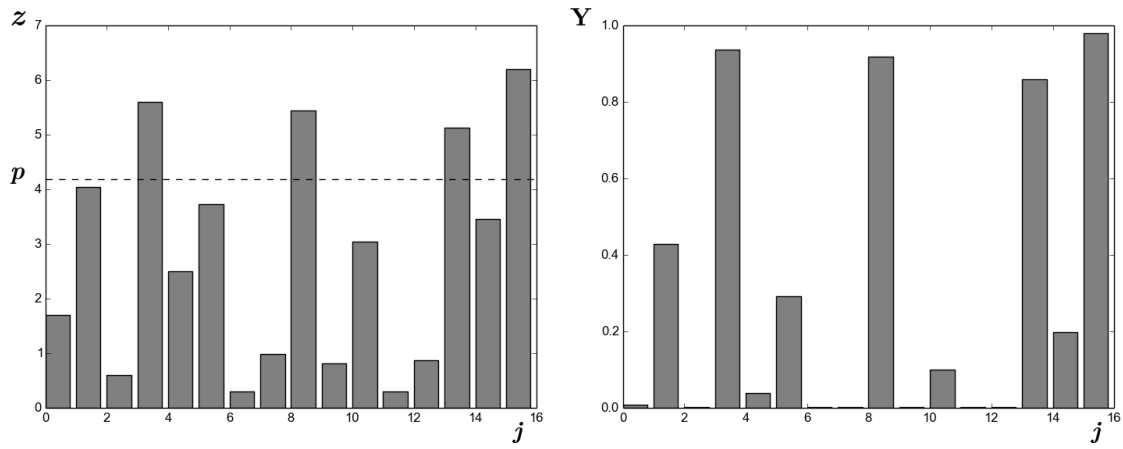

Figure 1. Stimulus and activation levels for 16 units with k=4.

4. DERIVATION OF THE LEARNING RULE

To perform the learning it is possible to calculate the corrections from an estimate of how much can be described about input X from the active units in the output. The simplest form consists of a linear approximation XX of the input computed from the output units Yj and the weights W.

$$\tilde{X}_i(Y, W) = \sum_j^N Y_j W_{ij}$$

The gradient of the corrections of the weights is determined by a learning rule derived from a cost function E that minimizes the sum of quadratic errors from the difference between the original input and the calculated approximation. In the expression of XX the terms Y and W were explicitly included to clearly specify their origin. In the same way, X and W will also be explicitly included in the expression of Y. In addition, as the cost is calculated over the whole dataset, a superscript indicates the corresponding instance, and a subscript indicates the

$$E(X^U, W) = \frac{1}{2} \sum_u \sum_i \left(X_i^u - \tilde{X}_i(Y(X^u, W), W) \right)^2$$

$$E_i^u(X^u, W) = \frac{1}{2} \left(X_i^u - \tilde{X}_i(Y(X^u, W), W \right)^2$$

$$\frac{\partial E_i^u(X^u, W)}{\partial W_{i,j}} = \frac{\partial E_i^u(X^u, W)}{\partial \tilde{X}_i(Y(X^u, W), W)} \cdot \frac{\partial \tilde{X}_i(Y(X^u, W), W)}{\partial W_{i,j}}$$

$$\frac{\partial E_i^u(X^u, W)}{\partial \tilde{X}_i(Y(X^u, W), W)} = \frac{1}{2} \cdot 2 \left(X_i^u - \tilde{X}_i(Y(X^u, W), W) \right)$$

$$\frac{\partial \tilde{X}_i(Y_j(X_i^u, W), W)}{\partial W_{i,j}} = Y_j(X^u, W)$$

$$\frac{\partial E_i^u(X^u, W)}{\partial W_{i,j}} = (X_i^u - \tilde{X}_i) \cdot Y_j$$

corresponding variable.

The resulting learning rule is identical to the Hebbian learning rule originally proposed by Oja [14]. This rule, used with linear activation units, is able to learn a transformation onto a space equivalent to that found by Principal Component Analysis (PCA). That is to say, even changing the way in which activation is obtained in the output layer, the corrections in the weights are computed in the same way.

This has special interest since sparse coding has a disadvantage when applied to an architecture for dimension reduction such as that of a typical autoencoder. It is generally estimated that, for a sparse coding to be effective, only 12% to 25% of the units must be activated at a time. For this to be achieved, an overcomplete coding is usually used, that is, the case where N>M [15]. For the following tests an overcomplete architecture will be used, trying to capture the characteristics that best represent the input data [16].

5. EXPERIMENTAL RESULTS

The tests performed consisted of comparing two of the most commonly used methods to produce a dispersed coding, the rectified linear units plus a penalty term in the cost function (ReLU+λL1), and the selection of the largest k values (SparseTopK), with the technique proposed in this work, COding with Logistic Softmax Sparse UnitS (COLOSSUS).

All models were trained for an equal and fixed number of epochs. The results shown were obtained by averaging several independent runs. Tests for all methods were performed by varying the number of output units (N), which is the number of features, and the proportion of simultaneous active units (p) expected. The data set used consisted of 6×6 pixel patches, with values bounded between 0 and 1, from the MNIST set. The results analyzed are the convergence speed, estimated from the training error, the test error, from the MSE, and the coding sparsity, from the activation level obtained by the L1 norm.

Table 1. MSE loss for training data after 20 epochs.

Method	**N=48 p=12.5%**	**N=64 p=12.5%**	**N=72 p=12.5%**
ReLU+λL1	11.118	10.217	9.981
SparseTopK	29.754	19.698	16.935
COLOSSUS	6.903	4.564	4.012
	N=48 p=25%	**N=64 p=25%**	**N=72 p=25%**
ReLU+λL1	10.787	9.833	9.210
SparseTopK	9.936	7.152	6.161
COLOSSUS	5.336	3.987	3.831
	N=48 p=50%	**N=64 p=50%**	**N=72 p=50%**
ReLU+λL1	9.253	8.580	8.114
SparseTopK	4.346	3.426	3.101
COLOSSUS	4.478	3.721	3.387

Table 1 shows the training loss taken at epoch 20, for 60,000 data instances, using the Mean Square Error (MSE). This measure gives an idea of how quickly different methods can converge to a good solution under different conditions. In the case of a higher level of dispersion, the SparseTopK method seems to converge marginally faster, but the proposed method offers consistently good results in any situation.

Table 2. MSE loss for testing data after training.

Method	**N=48 p=12.5%**	**N=64 p=12.5%**	**N=72 p=12.5%**
ReLU+λL1	0.00136	0.00131	0.00123
SparseTopK	0.00804	0.00516	0.00458
COLOSSUS	0.00181	0.00122	0.00111

	N=48 p=25%	N=64 p=25%	N=72 p=25%
ReLU+λL1	0.00138	0.00123	0.00120
SparseTopK	0.00263	0.00188	0.00162
COLOSSUS	0.00144	0.00106	0.00105

	N=48 p=50%	N=64 p=50%	N=72 p=50%
ReLU+λL1	0.00123	0.00114	0.00107
SparseTopK	0.00117	0.00092	0.00083
COLOSSUS	0.00121	0.00102	0.00091

Table 2 compares the evaluation errors, also calculated using MSE, with fully trained models, over 10,000 instances never seen during training. This is the measure that is usually shown to compare efficacy, but as it can be seen, except for small differences, the performance is similar in all cases.

Table 3. Coding sparsity measured as L_1 norm.
The expected active units are denoted by k.

Method	N=48 k=6	N=64 k=8	N=72 k=9
ReLU+λL1	6.845	6.130	6.142
SparseTopK	14.347	16.619	17.290
COLOSSUS	11.516	14.764	16.605

	N=48 k=12	N=64 k=16	N=72 k=18
ReLU+λL1	7.510	6.777	6.459
SparseTopK	19.928	24.028	26.045
COLOSSUS	14.256	18.618	21.705

	N=48 k=24	N=64 k=32	N=72 k=36
ReLU+λL1	9.593	8.802	8.142
SparseTopK	39.559	45.221	45.808
COLOSSUS	23.907	32.037	35.882

Finally, in Table 3 we try to show how sparse the coding is. For this purpose, the L1 norm over the activation at the output of the models is used as a measure. This value alone can be difficult to interpret, so instead of specifying the proportion of active units (p), the amount of active units (k) to which it is targeted is included. As, in the first case, under more restricted conditions ReLU units offer better results, but the operation of the proposed method is consistently better in a greater variety of cases.

6. CONCLUSIONS

Methods such as ReLU take advantage of the efficiency of max function calculation, but the addition of the L1 penalty in the cost function makes training more difficult and the results are not always reliable. The selection of a portion of maximum activation values requires a more complicated implementation where all the efficiency gained by using the easily computable max function is lost. The technique proposed in this work not only avoids these problems, since it can be trained with a very simple learning rule, and can be effectively calculated as the logistics of a softmax multiplied by a constant, but also offers at least equivalent results and in many cases better, than the other techniques, both in convergence speed, feature extraction, and coding sparsity.

REFERENCES

[1] Vincent, Pascal and Larochelle, Hugo and Lajoie, Isabelle and Bengio, Yoshua and Manzagol, Pierre-Antoine, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion", Journal of machine learning research (2010), 3371-- 3408.

[2] Mikolov, Tomas and Sutskever, Ilya and Chen, Kai and Corrado, Greg S and Dean, Jeff, "Distributed Representations of Words and Phrases and their Compositionality", Curran Associates, Inc. (2013), 3111--3119.

[3] Sutskever, Ilya and Hinton, Geoffrey, "Learning multilevel distributed representations for high-dimensional sequences" (2007), 548--555.

[4] Samuel Kaski and Teuvo Kohonen, "Winner-take-all networks for physiological models of competitive learning", Neural Networks (1994), 973 - 984.

[5] Rolls, Edmund T and Treves, Alessandro, "The relative advantages of sparse versus distributed encoding for associative neuronal networks in the brain", Network: computation in neural systems (1990), 407--421.

[6] Rehn, Martin and Sommer, Friedrich T, "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields.", J Comput Neurosci (2007), 135- 46.

[7] Bridle, John S., "Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recogni...", Springer Berlin Heidelberg (1990), 227-- 236.

[8] Donoho, David L. and Elad, Michael, "Optimally sparse representation in general (nonorthogonal) dictionaries via L1 minimization", Proceedings of the National Academy of Sciences (2003), 2197--2202.

[9] Gregor, Karol and LeCun, Yann, "Learning Fast Approximations of Sparse Coding" (2010), 399-406.

[10] Lee, Honglak and Battle, Alexis and Raina, Rajat and Ng, Andrew Y, "Efficient sparse coding algorithms" (2007), 801--808.

[11] Coates, Adam and Ng, Andrew Y, "The importance of encoding versus training with sparse coding and vector quantization" (2011), 921--928.

[12] Hinton, Geoffrey E, "Learning multiple layers of representation.", Trends Cogn. Sci. (Regul. Ed.) (2007), 428-34.

[13] Rumelhart, David E. and Hinton, Geoffrey E. and Williams, Ronald J., "Learning representations by back-propagating errors", nature (1986), 533.

[14] Oja, Erkki, "Principal components, minor components, and linear neural networks", Neural networks (1992), 927--935.

[15] Olshausen, Bruno A and Field, David J, "Sparse coding with an overcomplete basis set: A strategy employed by V1?", Vision research (1997), 3311--3325.

[16] Porrill, John and Stone, James V, "Undercomplete independent component analysis for signal separation and dimension reduction", Citeseer (1998).