MATRIX EQUATIONS IN DEEP LEARNING RESOLUTION FOR M DATA HAS N PARAMETERS

1Tshibengabu T Yannick, 2 Mbuyi M Eugene, 3 Batubenga M Jean-Didier

[1] *Masters student, Department of mathematics and computer science, University of Kinshasa, Drc;*

[2] *Ordinary Professor, Department of mathematics and computer science, University of Kinshasa, Drc;*

[3] *Associate Professor, Department of mathematics and computer science, University of Kinshasa, Drc.*

***Email**: {[1] yannick.tshibengabu@unikin.ac.cd , [2] eugenembuyi@gmail.com, [3] jude.batubenga@unikin.ac.cd}*

Abstract

This article on the vectorization of learning equations by neural network aims to give the matrix equations on [1-3]: first on the model Z [8, 9] of the perceptron [6] which calculates the inputs X , the W weights and the bias, second on the quantization function [10] [11], called the loss function [6, 7] [8]. and finally the gradient descent algorithm for maximizing likelihood and minimizing Z errors [4, 5] that can be applied in the classification of emotions by facial recognition.

Key Words - Key Words - Machine learning, cost function, gradient descent, the perceptron, vectorization.

1. Introduction

The important equations are based on the neural network, more precisely in the perceptron. Our concern in this writing is to implement a generalized solution based on the model.

$Z = w_1 . x_1 + w_2 . x_2 + b$, the cost function $L = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} * \log\left(a^{(i)}\right) + \left(1 - y^{(i)}\right) * \log(1 - a^{(i)})$ and on the descent of gradients $w_1 = w_1 - \alpha\frac{\partial L}{\partial w_1}$, $w_2 = w_2 - \alpha\frac{\partial L}{\partial w_2}$, $b = b - \alpha\frac{\partial L}{\partial b}$.

These different equations will have to be vectorized to have not only for 2 types of emotions with 2 parameters but to bring out the matrix equations of 8 (P) emotions with K parameters each. Our article returns to a review of the literature, the methodology used and finally the presentation of the experimental results.

2. Machine learning equations

2.1. The model

The perceptron model is a basic unit of neural networks [2], it is a binary classification model, capable of linearly separating 2 classes of points by a decision boundary, including positive and negative emotions. This model can be expressed by the following equation [13]:

$Z(x_1, x_2) = w_1 . x_1 + w_2 . x_2 + b$,

To improve the model, it is necessary to accompany this model with a probability [14], the further the face of an individual is from the decision boundary [15], the more it will be obvious (probable) that it belongs to a class determined, and the logistic activation function is a sigmoid function [16] of the form: $Z(a) = \frac{1}{1 + e^{-z}}$ which allows to convert the output Z into a single probability which follows Bernoulli's law [17], that of a face belonging to a single class, later in the CNN learning, we use a function ReLU which is defined as [3]: $f(x) = \max(0, x)$. Si $x < 0$, so $f(x) = 0$. Si $x \geq 0$, so $f(x) = x$ and which increases the chances of the network converging and does not cause saturation of the neurons unlike to the two functions tanh and sigmoid. With x_1, x_2 data associated with weight w_1, w_2 and an additional coefficient b, called the Bias.

The objective is to adjust the parameters b and W to have an efficient model that makes the smallest errors between the outputs a(Z) and the real data Y [16, 18].

2.2. The cost function [18]

A function that allows to quantify the errors made by the model in a classification, which allows to measure the distances between the outputs a (Z) and the data Y, with the aim of maximizing the likelihood L by minimizing the function –log (L). This function is of the form:

$$L = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} * \log\left(a^{(i)}\right) + \left(1-y^{(i)}\right) * \log(1-a^{(i)})$$

2.3. The Gradient Descent Algorithm [19]

Starting from our model Z and the function L, we need an automatic mechanism which consists in adjusting the parameters W and b of the function to minimize the errors of the model, that is to say to minimize the function cout (Log LOSS) hence the need to determine how this function varies according to the different parameters. The calculation of the gradient is simply the derivative of the function cout and is carried out according to the form below [6, 16]: $w_1 = w_1 - \alpha\frac{\partial L}{\partial w_1}$, $w_2 = w_2 - \alpha\frac{\partial L}{\partial w_2}$, $b = b - \alpha\frac{\partial L}{\partial b}$.

We have : $\frac{\partial L}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)}).x_1$

for the derivative $\partial w1$, $\frac{\partial L}{\partial w_2} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)}).x_2$ for ∂w_2

Et $\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$ for the b.

The vectorization of equations [20, 21] Machine learning by neural network requires the use of certain elementary operations that we recover from matrix equations: addition and subtraction, transpose and multiplications [22].

3. DataSet with 2 data types of 2 parameters

Y=0 (Negative emotions), Y=1 (Positive emotions).

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix} \in \mathbb{R}^{mxn} \text{ et } y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^{mx1}$$

We have a convention, where m is the number of data and n: number of variables in our DataSet n (positive emotion and negative emotion).

3.1. The X to Z Matrix Transformation

$$Z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + b \quad , \quad Z = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} =$$

$$\begin{bmatrix} w_1 x_1^{(1)} + w_2 x_2^{(1)} + b \\ w_1 x_1^{(2)} + w_2 x_2^{(2)} + b \\ \vdots \\ w_1 x_1^{(m)} + w_2 x_2^{(m)} + b \end{bmatrix}$$

This gives:

$$= \underset{X}{\begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix}} \cdot \underset{W}{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}} + \underset{b}{\begin{bmatrix} b \\ b \\ \vdots \\ b \end{bmatrix}}$$

X (m, 2), W (2,1), b (m, 1) and if we think of rewriting in matrix form, we have Z= X*W +b

$$b = \begin{pmatrix} b \\ b \\ \vdots \\ b \end{pmatrix} \in \mathbb{R}^{mx1} , W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^{nx1} , X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix} \in \mathbb{R}^{mxn}$$

whence b can be a real number b, such that b = b ∈ $\mathbb{R}$, with the principle of broadcasting, we also transform the equation a into a matrix.

$$a = \frac{1}{1+e^{-Z}}$$

The function can be written: $a^{(i)} = \sigma\left(Z^{(i)}\right) = \frac{1}{1+e^{-Z(i)}}$

Thus, we can replace each term by: $A = \begin{bmatrix} a^{(1)} \\ a^{(2)} \\ \vdots \\ a^{(m)} \end{bmatrix} =$

$\begin{bmatrix} \sigma(z^{(1)}) \\ \sigma(z^{(2)}) \\ \vdots \\ \sigma(z^{(m)}) \end{bmatrix}$ we can go out σ ;

$$= \sigma \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(m)} \end{bmatrix}$$

$= \sigma(Z)$

where Z is vector.

3.2. Vectorization of the cost function

A cost function makes it possible to calculate the global error by summing the individual errors of a and y in order to make a comparison between A and the vector y.

$$L = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} * \log\left(a^{(i)}\right) + \left(1-y^{(i)}\right) * \log(1-a^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m} \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} * \log(\begin{bmatrix} a^{(1)} \\ a^{(2)} \\ \vdots \\ a^{(m)} \end{bmatrix}) + \left(1 - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}\right) * \log(1 - \begin{bmatrix} a^{(1)} \\ a^{(2)} \\ \vdots \\ a^{(m)} \end{bmatrix})$$

$$= -\frac{1}{m}\sum_{i=1}^{m} \begin{bmatrix} y^{(1)} * \log\left(a^{(1)}\right) + \left(1-y^{(1)}\right) * \log\left(1 - a^{(1)}\right) \\ y^{(2)} * \log\left(a^{(2)}\right) + \left(1-y^{(2)}\right) * \log\left(1 - a^{(2)}\right) \\ \vdots \\ y^{(m)} * \log\left(a^{(m)}\right) + \left(1-y^{(m)}\right) * \log\left(1 - a^{(m)}\right) \end{bmatrix}$$

Who becomes a vector (m, 1)

From where $= -\frac{1}{m}\ \left[y^{(1)} * \log\left(a^{(1)}\right) + \left(1-y^{(1)}\right) * \log\left(1-\ a^{(1)}\right) + y^{(1)} * \log\left(a^{(1)}\right) + \left(1-y^{(1)}\right) * \log\left(1-\ a^{(1)}\right) + y^{(m)} * \log\left(a^{(m)}\right) + \left(1-y^{(m)}\right) * \log\left(1-\ a^{(m)}\right)\right]$

Which in principle must give a real number.

3.3. Vectorization of Gradient Descent functions.

The model has three parameters, namely $w_1, w_2, b.$ where $w_1, w_2 \in W$,

$W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$;

$w_1 = w_1 - \sigma \frac{\partial L}{\partial w_1}$ (1) ;

$w_2 = w_2 - \sigma \frac{\partial L}{\partial w_2}$ (2) ;

$b = b - \sigma \frac{\partial L}{\partial b}$

taking (1) et (2)

$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_2 \\ w_2 \end{bmatrix} - \sigma \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix}$;

$W = W - \sigma \left[\frac{\partial L}{\partial W}\right]$;

$b = b - \sigma \frac{\partial L}{\partial b}$, b is a real number (not a vector) so the partial derivative de $\frac{\partial L}{\partial b}$ is also a real number.

Which gives the 2 equations to implement for the realization of the Descent of the gradient. $W = W - \sigma \frac{\partial L}{\partial W}$; $b = b - \sigma \frac{\partial L}{\partial b}$

3.4. Gradient vectorization

The $\frac{\partial L}{\partial W}$ et $\frac{\partial L}{\partial b}$

$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix}$, replace each line with its expression.

$= \begin{bmatrix} \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})\ x_1^{(i)} \\ \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})\ x_2^{(i)} \end{bmatrix}$

$$= \begin{bmatrix} (a^{(1)} - y^{(1)})x_1^{(1)} + (a^{(2)} - y^{(2)})x_1^{(2)} + \cdots + (a^{(m)} - y^{(m)})x_1^{(m)} \\ (a^{(1)} - y^{(1)})x_2^{(1)} + (a^{(2)} - y^{(2)})x_2^{(2)} + \cdots + (a^{(m)} - y^{(m)})x_2^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(m)} \end{bmatrix} * \left(\begin{bmatrix} a^{(1)} \\ a^{(2)} \\ \vdots \\ a^{(m)} \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \right)$$

X^T (2, m), A(m,1), y (m,1)

$\frac{\partial L}{\partial W} = \frac{1}{m} X^T (A * y)$

$\frac{\partial L}{\partial b}$, b being a real number and not a vector, its derivation also gives in real.

$\frac{\partial L}{\partial b} = \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})$

$= \frac{1}{m} \sum a^{(1)} - y^{(1)} + a^{(2)} - y^{(2)} + \cdots + a^{(m)} - y^{(m)}$;

$\frac{\partial L}{\partial b} = \frac{1}{m} \sum(A - y)$

3.5. Vectorization on a DataSet of P data of K parameters

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_p^{(2)} \\ \vdots & \vdots & \ldots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \ldots & x_p^{(m)} \end{bmatrix} \in \mathbb{R}^{mxp},\ W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} \in \mathbb{R}^{kx1}$$

On aura:

$$W = W - \alpha \frac{\partial L}{\partial x} \Rightarrow \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_k} \end{bmatrix};$$

Avec

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})\ x_1^{(i)} \\ \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})\ x_2^{(i)} \\ \vdots \\ \frac{1}{m} \sum_{i=1}^{m}(a^{(i)} - y^{(i)})\ x_p^{(i)} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} x_1^{(1)}\ x_1^{(2)} & & x_1^{(m)} \\ x_2^{(1)}\ x_2^{(2)} & & x_2^{(m)} \\ & \vdots & \\ x_P^{(1)}\ x_P^{(2)} & & x_P^{(m)} \end{bmatrix} * \left(\begin{bmatrix} a^{(1)} \\ a^{(1)} \\ \vdots \\ a^{(m)} \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \right);$$

$\frac{\partial L}{\partial x} = \frac{1}{m} X^T(A - y)$.

3.6. Generalized models

Table I: Summary of matrix equations

equations	Description
The perceptron model	$Z = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix} . \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} b \\ b \\ \vdots \\ b \end{bmatrix}$ X W b
Activation function	$a = \sigma(Z)$
For a cost function	$L = -\frac{1}{m}\ \left[y^{(1)} * \log\left(a^{(1)}\right) + \left(1-y^{(1)}\right) * \log\left(1-\ a^{(1)}\right) + y^{(1)} * \log\left(a^{(1)}\right) + \left(1-y^{(1)}\right) * \log\left(1-\ a^{(1)}\right) + y^{(m)} * \log\left(a^{(m)}\right) + \left(1-y^{(m)}\right) * \log\left(1-\ a^{(m)}\right)\right]$
For the gradient:	$\frac{\partial L}{\partial x} = \frac{1}{m} X^T(A - y)$

4. Conclusion

Our approach being to bring out the material equations of three essential points in automatic learning by neural network, I quote the model Z with its inputs (X1…Xn) associated with the weight W, and the bias b, generalize a matrix for the cout function and the Gradient Descent algorithm, this procedure based on the vectorization of equations and the generalization of the latter can be applied to our dataset FER13 [23] dealing with facial emotional effects which may in the long run be used in the deep learning with any CNN architecture [24, 25] as appropriate.

5. Bibliographic references

1. Imane, R.G., S. KhawLa, and B. Kenza, Basic Emotion Recognition by Neural Networks: Application of deep Learning. 2021.
2. Malki, N. and T. Guerram, Automatic classification of texts by convolutional neural networks. 2019.
3. Guerradi, N. and H. Benkhelifa, Recognition of facial emotions by deep learning. 2019, جامعة غرداية.
4. Teroni, F. and A.K. ZIV,The shadows of the soul, thinking about negative emotions. 2011, Markus.
5. Krauth-Gruber, S., The regulation of emotions. Electronic Journal of Social Psychology, 2009. 4: p. 32-39.
6. Héritier, N.M. and I.B. Nephtali, The Gradient Backpropagation Algorithm in the Multilayer Perceptron: Basics and Case Study. International Journal of Innovation and Applied Studies, 2021. 32(2): p. 271-290.
7. Diagne, E.D.D., Discriminant analysis and multilayer perceptron - formal links and applications. 2019, University of Quebec at Trois-Rivières.
8. Lagnier, C., É. Gaussier, and F. Kawala, Modeling the user for the dissemination of information in social networks. Journal of Information Science and Technology-Series ISI: Information Systems Engineering, 2012. 17(3): p. 1-22.
9. Mercadier, Y., Automatic classification of texts by deep neural networks: application to the field of health. 2020, Montpellier University.
10. Morel, M. and T. Bänziger, The role of intonation in the vocal communication of emotions: test by synthesis Notebooks of the Institute of Linguistics of Louvain, 2004. 30(1-3): p. 207-232.
11. Granato, P., et al. Visual recognition of facial emotions in chronic schizophrenia. in Medico-psychological annals, psychiatric journal. 2009. Elsevier.
12. Logothetis, N.K. and D.L. Sheinberg, *Visual object recognition.* Annual review of neuroscience, 1996. 19(1): p. 577-621.
13. Zerzaihi, H. and F. Zarour, Image recognition by convolutional neural networks. 2020, University of Jijel.
14. Rachdi, N., Statistical learning and computer experiments: quantitative approach to risk and uncertainties in modeling. 2011, University of Toulouse, University of Toulouse III-Paul Sabatier.
15. Delaforge, A., et al., *EBBE-Text:* Visualization of the decision boundary of neural networks in automatic text classification. Knowledge Extraction and Management: Actes EGC'2021, 2021.
16. Coudé, C., Neural networks in the field of image classification. 2017.
17. Biane, P., Quantum Bernoulli walks, in Probability Seminar *XXIV 1988/89*. 1990, Springer. p. 329-344.
18. Gelly, G.g., Recurrent neural networks for automatic speech processing. 2017, Paris-Saclay University (COmUE).
19. Gillot, P., et al., Stochastic Gradient Descent algorithms with parameter filtering for training deep convolutional networks.
20. Sanchez, E., S. Barro, and C. Regueiro. *Artificial neural networks implementation on vectorial supercomputers*. in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. 1994. IEEE.
21. Parizeau, M., *Réseaux de neurones.* GIF-21140 et GIF-64326, 2004. 124.
22. Oussar, Y., Wavelet networks and neural networks for static and dynamic process modeling. 1998, Pierre and Marie Curie-Paris VI University.
23. Abanoz, H. and Z. Çataltepe. *Emotion recognition on static images using deep transfer learning and ensembling*. in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. 2018. IEEE.
24. Umer, M., et al., *Fake news stance detection using deep learning architecture (CNN-LSTM).* IEEE Access, 2020. 8: p. 156695-156706.
25. Chao, Y.-W., et al. *Rethinking the faster r-cnn architecture for temporal action localization*. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.