

Regularization of a neural network for binary classification

Author: Andreu Bécares Mas

Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Advisor: M. Ángeles Serrano Moral

Neural networks are complex non-linear models used to learn representations of data with multiple levels of abstraction. In this work, we introduce the basic notions of a neural network model, explaining the training process and generalization capabilities. In particular, to avoid overfitting to the training data, we study the L2 regularization method. Finally, an application of 2D vector classification has been developed to illustrate the concepts.

I. INTRODUCTION

Neural networks are computing systems inspired by biological neural networks that learn to perform complex tasks identifying characteristics from the data they process. They play an important role in the field of machine learning, more precisely in the area of supervised learning, which comprises the tasks of learning complex functions that map an input to an output based on example input-output pairs.

A neural network is based on a collection of neurons stacked into layers that transmit signals from a layer to the next depending on a activation value determined by a non-linear function. Stacking these non-linear modules can transform a representation at one level into a more abstract representation in the next level. With the composition of these transformations, very complex functions can be learned. This is the concept behind Deep Learning, building networks with many layers between the input and the output of the network.

Deep neural networks take advantage of the property that natural signals are made of hierarchies, for example in images, where parts can be decomposed into more local details, or in text, where words form sentences and arguments.

With the growth of Big Data in recent years, neural networks have emerged as a very powerful tool for pattern recognition. They are being used for a variety of tasks like image recognition, speech recognition, social network filtering, general game playing and many other domains like drug discovery and genomics.

The motivation of this work is to study Feed-forward neural networks, the most simple neural network structure where the information moves in one direction, from an input layer to an output layer. We will study the mathematical functions related with the neurons, the meaning of a cost function and the process behind the backpropagation algorithm that optimizes the network. To further understand the network training and generalization of the extracted features to new data, the topic of overfitting will be discussed as well as a method of regularization to avoid it. Lastly, to put in practice the theory, we will present a simple neural network application of binary classification with a set of 2D vectors associated with a binary label.

II. METHODS

In this section we will explain different methodologies related with neural networks functioning. We will start by describing the neural network model and its parts. Next, the method of L2 regularization will be explained and how it changes the network to reduce overfitting to the training data. At the end, we will do an introduction to the application model of binary classification.

A. Neural network model

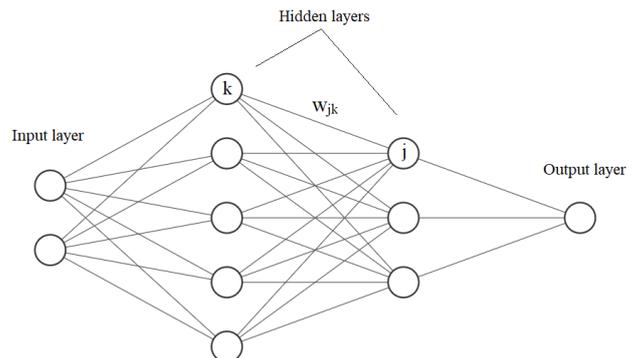


FIG. 1: Structure of a feed-forward neural network with two hidden layers. The input layer has two input signals and the output layer returns a single value. Each node represents a neuron, which performs an operation based on all the outputs from the previous layer and returns a value to the next layer.

A neural network consists of neurons that are ordered into layers (see Figure 1). The first layer is called the input layer, the last is the output layer and the layers in between are the hidden layers. Each neuron in a particular layer is connected with all the neurons in the next layer.

The input layer gathers the data that is to be analyzed. This requires a transformation of the data into a vector where each component corresponds to a input neuron. For example, in image recognition, the three RGB ma-

trices of pixels are reshaped into a vector column. This way, each component from the vector represents some intensity value of a color in a pixel.

At the end, the outcome of the output layer is used to make a prediction. If it is a single value it can be compared to a threshold to decide between different classes. In other cases, where there are many output neurons, the highest value is chosen as the predicted class.

The neurons, the units of the network, represent a function that takes a vector of k input features $x = (x_1, x_2 \dots x_k)$ and returns a scalar output $a(x)$. This function can be split into two parts: a linear operation and a non-linear transformation. The linear operation is a dot product with a set of weights $w_{jk} = (w_1, w_2 \dots w_k)$ that reflect the importance of the different k inputs and a correction with a bias b_j . This is then introduced to the non-linear function, also called activation function, that limits the amplitude of the output. For a neuron j :

$$z_j = \mathbf{w}_{jk} \cdot \mathbf{x} + b_j, \quad a_j = f(z_j) \quad (1)$$

Activation function

The activation function introduces non-linearity in the network that allows it to adapt to complex patterns. There are different choices for activation functions. In the beginning of the field of neural networks, functions like step-functions, sigmoids and hiperbolic tangents were used. Later, it would be acknowledged that these had a negative impact on the training phase because of their saturation for high inputs [1]. Nowadays, ReLUs (Rectified Linear Units) are the most common practice as their gradients stay finite even with large inputs (see figure 2).

However, some of the before-mentioned functions are still used in the output layer to limit the amplitude of the output in a desired range, for example, the sigmoid function limits the output between 0 and 1.

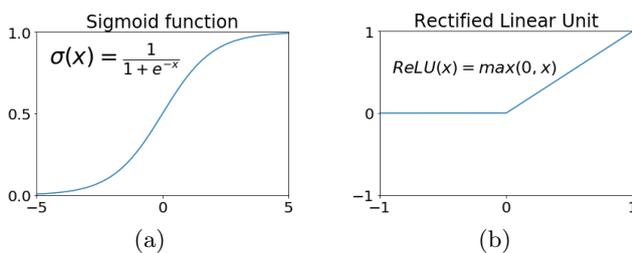


FIG. 2: Sigmoid and ReLU activation functions. They limit the amplitude of the output depending on the input value. The sigmoid saturates with high inputs making training harder.

Training and prediction

A neural network can work either in training or in prediction mode. For each mode there has to be a unique set of data, but all data has to come from the same distribution. The training data is used for optimizing the network and the test data to check its performance, so the test set is usually much smaller.

In order to train a neural network to make accurate predictions from an input we have to compute a function, called the cost function, that measures the error between the output scores and the desired pattern of scores. To reduce this error, on each iteration the network modifies its internal adjustable parameters: the weights and the biases. To properly adjust them, the learning algorithm computes a gradient for each parameter depending on its location on the network, indicating how the error would change with a variation of the parameter. The parameters are then updated on the opposite direction of the gradient to move the error towards a minimum of the cost function in next iterations. This process is called gradient descent, and is the basis of the algorithm of backpropagation that will be explained later.

The proceeding of a training iteration is as follows: at the start, once an input has been given to the network, random values are set to the weights and the biases are initialized as zeros. These different starting points allow the network to differentiate between all the neurons in a layer. The training then proceeds in two phases:

- In the forward phase, the parameters of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. The output depends on the activation values of the neurons, affected by the inputs and parameters.
- In the backward phase, an error signal is produced by comparing the output of the network with a desired response. With backpropagation, the gradients of the error signal with respect to the parameters are propagated through the network, again layer by layer, but in the backward direction. This way, successive adjustments are made to the parameters of the network.

After finishing the iterations of training, the updated parameters are used to make predictions. This is done by giving to the network the test data set and comparing the resulting scalar outputs to a threshold, without using the cost function or backpropagation. In binary classification this threshold is:

$$\hat{y} = \begin{cases} 1, & \text{if } a^L \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Where a^L is the activation value from the last layer and \hat{y} is the prediction. Other classification tasks require different methodologies.

Cost function

The cost function that compares the outputs of a network with the true labels that are given in supervised learning depends on the classification task. In binary classification, used later in this work, the network returns predictions that are either 0 or 1. As a consequence, the output of the last layer has the probability

$\hat{y} = p(y = 1|a_L)$ that in an iteration the input data will be predicted to be in category 1.

To define the cost function we use maximum likelihood estimation [2]. Considering a dataset $D = \{(y_i, x_i)\}$ with binary labels $y_i \in \{0, 1\}$, the likelihood of observing the data is:

$$P(D|w) = \prod_{i=1}^n (\hat{y}_i)^{y_i} (1 - \hat{y}_i)^{1-y_i} \quad (3)$$

Where y_i is the sigmoid function. The cost function is the negative of the log-likelihood, as we want to minimize the cost function instead of maximizing the likelihood:

$$E(w, b) = - \sum_{i=1}^n y_i \log \hat{y}_i(w, b) + (1 - y_i) \log [1 - \hat{y}_i(w, b)] \quad (4)$$

Backpropagation algorithm

In supervised learning we make use of gradient descent to optimize the cost function, updating the weights and biases to move in the direction where the cost function is reduced.

The calculation of the gradient in each layer is achieved with the Backpropagation algorithm. This algorithm can be seen as a propagation of an error backwards into the network using the chain rule of partial differentiation.

In a hidden layer, we can compute the derivative of the cost function E with respect to the linear operation z_j^l of the j -th neuron in the l -th layer as [2]:

$$\frac{\partial E}{\partial z_j^l} = \left(\sum_k \frac{\partial E}{\partial z_k^{l+1}} w_{kj}^{l+1} \right) f'(z_j^l) \quad (5)$$

Which takes account of the error from the next layer $l+1$ and the expression $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$ that relates the input of a neuron to an output of a previous layer. The derivatives with respect to the weights and biases are:

$$\frac{\partial E}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l} a_k^{l-1}, \quad \frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial z_j^l} \quad (6)$$

Where for the bias $\partial b_j^l / \partial z_j^l = 1$.

Once the equations in (6) are computed, in each iteration the weight and bias update is done as follows:

$$w'_{jk} = w_{jk} - \alpha \left(\frac{\partial E}{\partial w_{jk}^l} \right), \quad b'_j = b_j - \alpha \left(\frac{\partial E}{\partial b_j^l} \right) \quad (7)$$

The parameter α is called the learning rate. It is fixed during training and affects the size of the weight and bias update. If too small, training may take more iterations, while if too big, the cost may oscillate up and down without converging.

B. Overfitting and regularization

When the training process takes too many iterations, the network can adapt too much to the training data

and be less able to generalise its features with new data. This is called overfitting, when the network gives perfect results from the training set but fails for examples in the test set.

The basic condition to avoid overfitting is having a sufficiently large set of training samples that is representative of all the cases interested to extract features from. When this is not possible, there are different approaches to avoid overfitting, of which we will study L2 regularization. In networks with high numbers of connections between layers, overfitting can come from an uneven distribution of the weight values. The more we train a network, the bigger the weights get, specializing to the training data. This makes the output unstable, as minor variations on the inputs can result in big changes of the output.

In L2 regularization a term is added to the cost function (equation 4), so that while looking to minimize its value, attention is put into having a more balanced consideration of neuron inputs. The term is the sum of the squared euclidean norm of the weight matrices from each layer. An additional parameter λ is added to control the strength of the regularization.

$$E_{L2}(w) = E(w) + \frac{1}{m} \frac{\lambda}{2} \sum_{l=1}^L \|W^l\|^2 \quad (8)$$

Where m is the size of the training data set and is used as a scaling factor. The addition of the L2 term is then included in the weight update and results in much smaller weights across the entire model.

C. Application: 2D vector classification

We have developed an application of a neural network in Python that learns to classify 2D vectors with a binary label (color red or blue), so it is a binary classification problem.

The data is generated with code from *sci-kit learn* [4], an open source machine learning library in Python. It offers different distributions of data points for machine learning problems. The code allows adding Gaussian noise to the data points, a feature we will use to analyze overfitting.

The code of the neural network was developed with the knowledge gained from the online course *Neural Networks and Deep Learning* from Coursera [3]. The course is an introduction to Deep Learning and teaches how to program a neural network step by step in Python.

The network cost function and the backpropagation are implemented as they have been explained in the previous subsections. Once the parameters are trained, a uniform 2D grid of points is passed point by point into the network to return the boundary of separation of the two classes.

III. RESULTS

The neural network application has a similar shape to the example in figure 1. There are two input neurons on the first layer, where each one corresponds to a value of the x and y positions of a point. On the two hidden layers it has 20 and 10 neurons respectively. On the output layer there is a single neuron that returns a scalar between 0 and 1 for binary classification thanks to a sigmoid activation function. The hidden layers use ReLU activation functions. Other fixed parameter in the network will be the learning rate, which is $\alpha = 0.3$.

First, we train the network on data with little noise, with a value of 0.05σ of standard deviation. In this case, training for 200 iterations the two classes are distant and the network is able to separate both classes with a test accuracy of 100% as all points have been classified correctly.

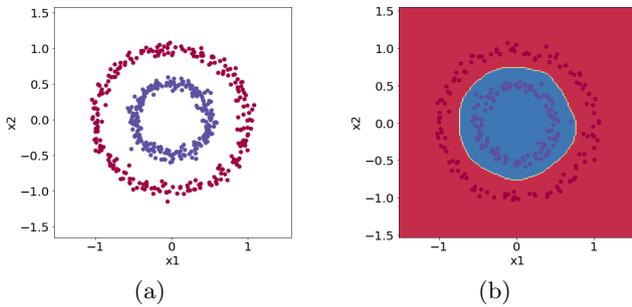


FIG. 3: Raw training data (a) and classification of the test data (b). The network manages to generalize well to the test data as the noise is not important enough and the classes are separate.

When introducing a noise of 0.2σ in the data the boundary of separation is not clear anymore. When we train the network for 20000 iterations, we start to see some overfitting effects (see figure 4). The network is adapting to the training data. The test data is generated with a different random seed, so the classification boundary doesn't generalize well. Implementing L2 regularization, the classification boundaries are simplified as a result of the constrain that the weights of the network now contribute to the cost function.

The accuracy of the network predictions in the four cases is shown in the table below, where the accuracy is obtained from the matches of the predictions with their corresponding true label averaged over the samples in the data set.

Mode	Acc. training (%)	Acc. test (%)
Standard	90.0	84.0
L2 regularization	89.0	87.6

TABLE I: Accuracy on the training and test sets with the network operating in standard mode and with L2 regularization.

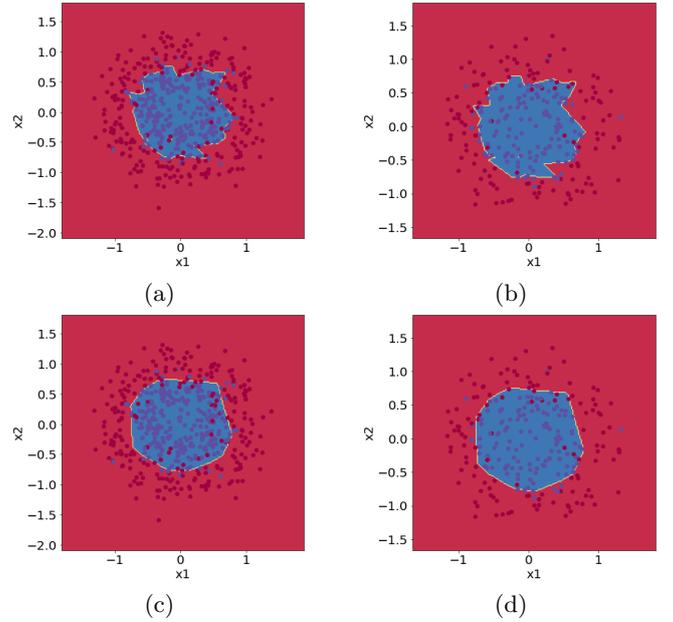


FIG. 4: Overfitted training set (a) and test set (b). The classification has adapted to specific inputs during training that are not present in the test data. In (c) and (d) we have included L2 regularization in the cost function with $\lambda = 1$.

In standard mode, the accuracy on the train set ends up being higher because the network adapts to specific training samples during the large training process trying to reduce the cost function even further. This gives high accuracy in training, but with test data it is not be able to generalise the features and the accuracy drops. With L2 regularization, training accuracy is reduced but test accuracy is boosted as a result of a more simple boundary. In general, what is wanted from neural networks is that they have high accuracy values in test accuracy, meaning that the network can make good predictions when considering new cases.

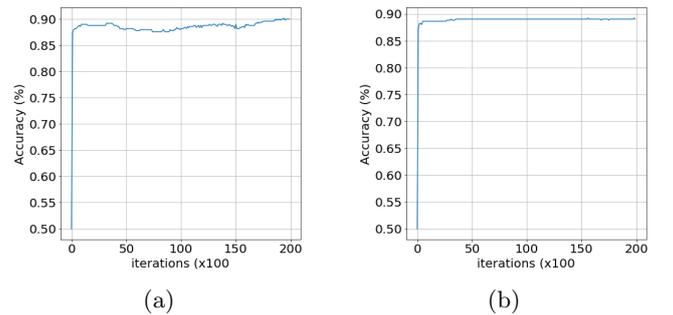


FIG. 5: Evolution of the accuracy during training in standard mode (a) and with L2 regularization (b).

The evolution of the accuracy during training (figure 5) for the standard mode can be irregular because the network adapts to inputs that reduce the cost function

but the new inputs that come later don't align with those corrections. With regularization, learning is more stable and the accuracy saturates at a lower value.

Another way of visualizing how regularization affects the network is by measuring the size of the weights (see figure 6). In our network structure we have a total of 250 weights. To view their sizes in a 3D plot we stacked them into 25 and 20 on each horizontal axis.

In line with what we have seen before, after regularization the weights shrink significantly forming a much more even distribution and giving stability to learning but simplifying the model.

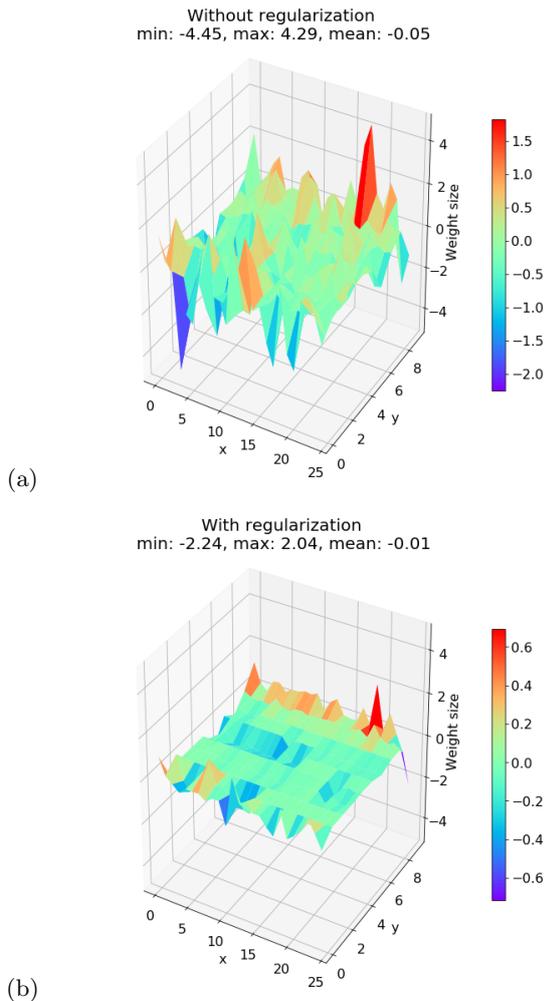


FIG. 6: Sizes of the weight matrix without regularization (a) and with L2 regularization (b).

IV. CONCLUSIONS

This study gives a brief insight into how artificial neural networks are trained in supervised learning problems, the structure of the networks and how neurons activate, the definition of a loss function and the concept of back-propagation. Furthermore, it helps identify the scenarios where a network is overfitting the data, how this affects the predictions and also its parameters, specially the weights of the connections.

Results have shown us that the inclusion of the L2 regularization term into the loss function has different consequences to the neural network performance. First, it simplifies the model, stopping it from optimizing the cost function to specific inputs during training. Also, it makes the model training phase to be smoother but slower, saturating at lower values of accuracy. However, the simplification of the model allows the predictions to be more valuable on the test data, as the model generalises better its features. Finally, we have measured how much the weights have been reduced with regularization. With these results, we can conclude that L2 regularization is an effective method for reducing overfitting.

This investigation opens a lot of questions for further work. For instance, learning more about other methods of regularization like early stopping or dropout that were not be included here. Also, the topic of the optimal shape of a network for a certain classification problem and the complexity that layers bring to the prediction would be interesting to be studied but this discussion was more centered around overfitting and regularization.

V. APPENDIX

If you are interested in reading the code of the application, it will be posted in my Github repository: <https://github.com/AndrewBM>

Acknowledgments

I would like to express my gratitude to my advisor, M. Angeles Serrano Moral for giving me the opportunity of studying a field of my interest and for her attention and guidance. I would also like to thank my family and friends for their support and encouragement over these years.

-
- [1] LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. *Deep learning*. Nature, 2015, vol. 521, no 7553, p. 436-439
- [2] MEHTA, P. et al, *A High-Bias, Low-Variance Introduction to Machine Learning for Physicists*, arXiv:

- 1803.08823. p. 29-30,45-52
- [3] Coursera course "Neural Networks and Deep Learning" by deeplearning.ai
- [4] Scikit-learn datasets: <https://scikit-learn.org/>