Making neural nets more accessible from first principles

Blanca Cano Camarero, Jose Ramón Dorronsoro Ibero [0], Juan Julián Merelo Guervós [1], Francisco Javier Merí de la Maza[2].

Abstract
The algorithm introduced in this paper allow neural network weights' initialization based on training data. The proposed method prevent from transfer learning and also simplifies neural network training, by computing their weights with the least error possible.

Keywords
Weight initialization — neuronal network — Optimization

[0] *Department of Computer Science, Universidad Autónoma Madrid, Madrid, España*
[1] *Department of Computer Architecture, University of Granada, Granada, España*
[2] *Department of Mathematical Analysis, University of Granada, Granada, España*

Contents

Introduction

We have tried to apply agile principles to the whole development process, from the more mathematical to the minimally viable product that was the developed code and this paper [1].

Since in 1988, Hornik, Stinchombe and White established that Multilayer Feedforward Networks are Universal Approximators [2] this tools have revolutionized artificial intelligence.

Backpropagation [3] and its variants are relevant methods to update neural network's weights. However one of its weak spots comes from being iterative algorithms, since an initial point is needed (ie a initial setting of the weights of the neuronal network). In order to determine this point there are some heuristics such as random initialization closed to null values or features transference.

In order to determine that initial state we propose a initialization method based on the training data.

To introduce our method we are going to use One Layer Feedforward Networks [2] , from now on we will refer to them as Neural Networks. Moreove, we show how this algorithm improve random initiallization.

State of the art

Algorithm

Let models Neural Networks the elements of the following functional space:

For $X \subseteq \mathbb{R}^d, Y \subseteq \mathbb{R}^s$.

$$\mathscr{H}(X,Y) = \{h : X \longrightarrow Y / \quad h_k(x) = \sum_{i=1}^{n} \beta_{ik}\gamma(A_i(x))\}. \tag{1}$$

Where h_k, $k \in \{1,\dots,s\}$ is the k-projection of h, $n \in \mathbb{N}, \gamma$ an activation function [1], $\beta_{ik} \in \mathbb{R}$ and A_i is an affine function from $\mathbb{R}^d$ to $\mathbb{R}$.

Fixed the activation function γ, we can see a Neural Network h of n hidden units, $d \in \mathbb{N}$ entry dimension and s output dimension as

$$h : \mathbb{R}^d \longrightarrow \mathbb{R}^s, \tag{2}$$

$$h_k(x) = \sum_{i=1}^{n} \left(\beta_{ik}\gamma \left(\sum_{j=1}^{d} (\alpha_{ij}x_j) + \alpha_{0j} \right) \right). \tag{3}$$

determined by its params:

$$(A,B) \in R^{n\times(d+1)} \times R^{s\times n}. \tag{4}$$

$$A = (\alpha_{ij}) \text{ con } i \in \{0,\dots d\},\ j \in \{1,\dots n\}.$$
$$B = (\beta_{jk}) \text{ con } j \in \{1,\dots n\},\ k \in \{1,\dots s\}.$$

For our algorithm we will determine the value of (A,B) seeing them and the training data as an oversized system of linear equations.

[1] The definition of activation function will came in next sections

Algorithm description

Let be $h \in \mathscr{H}(\mathbb{R}^d, \mathbb{R}^s)$ with n hidden units and let $M \in R^+$ chosen conveniently[2].

1. Take randomly $p \in \mathbb{R}^{d+1} \setminus \{0\}$.
2. Let Λ be an empty set.
3. While $|\Lambda| < n$ repeat:
   - i. Pick randomly $(x,y) \in \mathscr{D}$.
   - ii. If x satisfies that for every $(z,w) \in \Lambda$

     $$p \cdot (x - z) \neq 0, \tag{5}$$

     then let $\Lambda \leftarrow \Lambda \cup \{(x,y)\}$.
4. Without loss of generality the elements of Λ

   $$\Lambda = \{(x_1,y_1),(x_2,y_2),\dots(x_n,y_n)\}$$

   are ordered by the following statement

   $$p \cdot x_1 < p \cdot x_2 < \dots p \cdot x_n. \tag{6}$$

5. Pick $(x_1,y_1) \in \Lambda$

   $$A_{1[1:d]} = 0p,$$
   $$A_{10} = M,$$
   $$B_{*1} = y_1.$$

   For $k \in \{1,\dots,n\}$

   $$A_{k0} = M - \frac{2M}{p \cdot (x_k - x_{k-1})}(p \cdot x_k),$$
   $$A_{ki} = \frac{2M}{p \cdot (x_k - x_{k-1})} p_i \quad i \in \{1,\dots d\},$$
   $$B_{*k} = y_k - \hat{y}_{k-1}.$$

   where $(x_k,y_k) \in \Lambda$, A_{ki} the element of the k row and i column, B_{*k} denoted the $k-$th row. $\hat{y}_{k-1}$ is determined by: If activation function is right and left asymptotic (for example *ramp function*), then

   $$\hat{y}_{k-1} = y_{k-1}. \tag{7}$$

   Otherwise, if activation function is only left asymptotic (for example *relu function*) then

   $$\hat{y}_{k-1} = h_{k-1}(x_k), \tag{8}$$

   where h_{k-1} is the neural network defined by known coefficient $A_{[0:k,*]}, B_{[*,0:k]}$.
6. (A,B) are the matrix we searched for.

Determine $\hat{y}_{k-1}$

Some values for M

The value of M is determined by the chosen activation function γ.

If the activation function is left and right asymptotic, the selected M should verifies

For every $K \geq M \geq 0$

$$\gamma(K) = 1 \tag{9}$$
$$\gamma(-K) = 0. \tag{10}$$

If the activation function is left asymptotic $M \in \mathbb{R}^+$ should verify

$$\gamma(M) = 1 \tag{11}$$
$$\gamma(-M) = 0. \tag{12}$$

Could be any real value bigger than the specified one:

Activation function	Minimum value of M
Ramp function	1
Cosine Squasher	$\frac{\pi}{2}$
Indicator function of 0	0
Hardtanh	1
Sigmoid	10 (for an error smaller than 10^{-5})
tanh	7 for an error smaller than 10^{-5})

Table 1. Minimum value of M for the algorithm depending of the chosen activation function.

Experiments

In order to show the improvement of the initialization we are going to create the following experiment:

For a fixed data set $\mathscr{D}$, we split it in three set:

- Train data set, T.
- Error in train data set, E_{in}.
- Test data set, E_{out}.

Once E_{in}^{Init} for our algorithm is computed, for a bathsize b, we count the minimum number of epochs of backpropagation needed in order to achieve $E_{in}^{Init} \geq E_{in}^{b}$.

Datasets

From Kaggle House Price Prediction `https://www.kaggle.com/datasets/shree1992/housedata`

Further works

Classification problems

Even though, as neurons are increased the error decreased, for classification problems where the output should be an exact class the Neural Network should be compose by a classification function.

Add photo.

[2] See subsection *Some values for M*

Symbolic equation for multilayer neural networks

The equations could be generalized for any neural architecture by symbolic calcs.

Reducing noise for tails

For no asymptotic activation functions the internode values x between the selected values x_k, x_{k+1}

$$p \cdot x_k < p \cdot x p \cdot x_{k+1} \tag{13}$$

the higher is k, more neurons will be activated by x and the higher and less smoothed will be its predictions.

This phenomena could increase the error ans some algorithm modification may be needed.

Appendix

Theorem 1. *For each* $(x_k, y_k) \in \Lambda \subset \mathscr{D}$ *the neural network* $h \in \mathscr{H}(\mathbb{R}^d, \mathbb{R}^s)$ *defined by the algorithm satisfy that*

$$h(x_k) = y_k.$$

Proof. Let $n \in \mathbb{N}$ be the amount of hidden cells. The size of $\mathscr{D}$ is at least n. Let $p \in \mathbb{R}^d$ be a vector that satisfies: exist $\Lambda \subset \mathscr{D}$ that verify

$$p \cdot (x_i - x_j) \neq 0$$

for every different x_i, x_j from Λ. □

References

[1] Juan Julián Merelo Guervós. Agile (data) science: a (draft) manifesto. *CoRR*, abs/2104.12545, 2021.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[3] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.