Machine learning algorithms Neural Networks

Carlos Perales-González[1]

[1]Universidad Loyola Andalucía

2020-2021

Overview

Remember: Supervised Machine Learning

Generally speaking, a predictor f is a function with vector of features $\boldsymbol{x}_n \in \boldsymbol{X}$ as input and $y_n \in \boldsymbol{Y}$ as output / target.

$$f(\boldsymbol{x}_n; \theta) \approx y_n,$$

The training process is a minimization of the Error function f over dataset $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ where optimal parameters $\theta \in \Theta$ are tuned.

$$\begin{aligned} \min_{\theta \in \Theta} \quad & \text{Error}((f(\boldsymbol{x}_1; \theta), y_1), \ldots, (f(\boldsymbol{x}_N; \theta), y_N)) \\ \text{subject to:} \quad & \text{Model restrictions} \end{aligned} \tag{1}$$

ELM (I)

ELM is a type of Neural Network where the weights in the hidden layer are predefined [2].

- **Prediction**:

$$f(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})^{'} \beta \tag{2}$$

with

$$\boldsymbol{h}(\boldsymbol{x}) = (\phi(\boldsymbol{x}; \boldsymbol{w}_d, b_d),\ d = 1, \ldots, D), \tag{3}$$

and ϕ is the activation function, for example sigmoid,

$$\phi(\boldsymbol{x}; \boldsymbol{w}_d, b_d) = \frac{1}{1 + \exp(-(\boldsymbol{w}'_d \cdot \boldsymbol{x} + b_d))}. \tag{4}$$

- **Training**:

$$\min_{\beta \in \mathbb{R}^D} \|\boldsymbol{H}\beta - \boldsymbol{Y}\|^2 + C\|\beta\|^2 \tag{5}$$

One Hot Encoding

ELM can be adapted to classification problems using 1-of-J / One Hot Encoding / Label Binarization [1].

$$y_n \to \boldsymbol{t}_n$$

$$\boldsymbol{t}_{j,n} = \begin{cases} 1 & \text{if } j \text{ is the class of } n\text{-th pattern} \\ 0 & \text{otherwise} \end{cases}$$

It can be found in *sklearn* library as OneHotEncoder.

ELM (II)

By deriving $\frac{\partial \text{Error}}{\partial \boldsymbol{\beta}} = 0$, the solution(s) can be obtained,

$$\boldsymbol{\beta} = \left(\boldsymbol{H}'\boldsymbol{H} + C\boldsymbol{I}\right)^{-1}\boldsymbol{H}'\boldsymbol{Y}, \tag{6}$$

$$\boldsymbol{\beta} = \boldsymbol{H}'\left(\boldsymbol{H}\boldsymbol{H}' + C\boldsymbol{I}\right)^{-1}\boldsymbol{Y}. \tag{7}$$

Both solutions are correct. $\boldsymbol{\beta}$ is replaced in equation (2).

Program yourself!

- Open a Jupyter Notebook and load Boston dataset (regression) using *sklearn*.
- Program ELM as an object with *fit* and *predict* methods.
- Test it with RMSE metric.
- Program a grid search function.

One single hidden layer: perceptron (I)

The most basic architecture, yet allow to solve nonlinear problems.

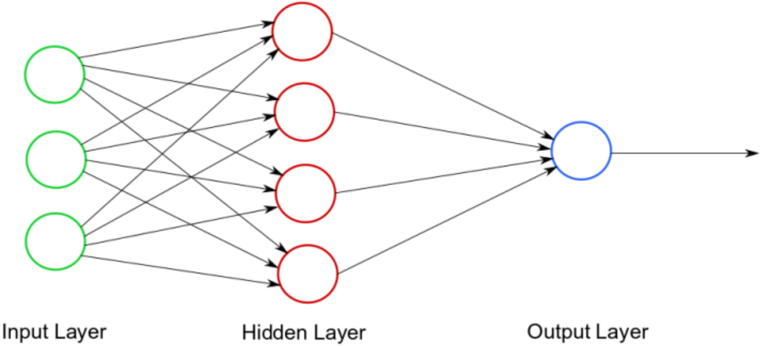


Figure: Perceptron for regression problems

Perceptron (II)

- Solve nonlinear problems.
- All the layers are tuned.
- Activation function in the hidden layer, adding non-linearity.

Perceptron (III)

The prediction is the flow of the information through the the Neural Networks.
The cost of the function can be defined by many loss functions.
Classical is Mean Squared Error.

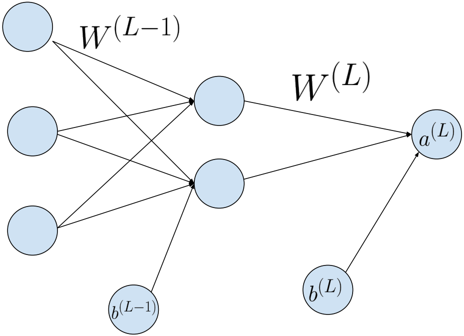


Figure: Perceptron for regression problems

Perceptron (IV)

$$Error = C = \frac{1}{N}\sum_{n=1}^{N}(f(\boldsymbol{x}_n) - y_n)^2 \tag{8}$$

- C is the cost of the neural network (the error).
- f is the prediction of the neural network.
- y_n is the real value of the n-th pattern.
- N is the number of elements in the dataset.

The output of the layer l, we will call it $a^{(l)}$. So prediction of the Neural Network is the output of the last layer, $a^{(L)}$. The activation of the nodes is denoted as $\sigma(z)$. There are several activation functions, classical is sigmoid, since its output is between 0 and 1 and its derivative is easy to calculate. The input of the first layer, $a^{(0)}$, is the vector of features, $\boldsymbol{x}$.

The outputs of each layer, a, are activated through σ. The product of the weights plus the bias is called z. This z is activated, $\sigma(z)$.

$$
\begin{aligned}
a^{(L)} &= \sigma(z^{(L)}) && (9)\\
\sigma_{sigmoid}(z) &= \frac{1}{1+e^{-z}} && (10)\\
\sigma'_{sigmoid}(z) &= \sigma(z)(1-\sigma(z)) && (11)\\
z^{(L)} &= w^{(L)}a^{(L-1)} + b^{L} && (12)\\
z^{(L-1)} &= w^{(L-1)}\boldsymbol{x}_n + b^{(L-1)} && (13)
\end{aligned}
$$

Perceptron (V)

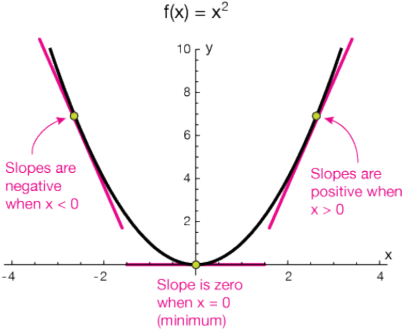
Classic solver is *backpropagation*, which improves the solution in several iterations $r = 1, 2, \ldots$. It is based on Gradient Descent minimization

$$W^{(l)}_{(r)} = W^{(l)}_{(r-1)} - \eta \frac{\partial C}{\partial w}\Big|^{(l)}_{(r-1)} \quad (14)$$

- subindex (r) is the iteration.
- superindex (l) is the layer.
- η is the learning rate. It is an hyper-parameter.

Perceptron (V)

Derivative of a convex function is 0 in its maximum / minimum. Graphically, it can be seen,



Perceptron (VI)

The η hyper-parameter control the learning step.

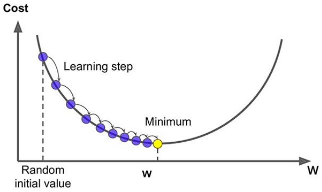


Figure: Learning step

Backpropagation (I)

Let's call C to the error from Equation (8). The derivate $\frac{\partial C}{\partial W^{(L)}}$ can be found by chain rule.

$$\frac{\partial C}{\partial W^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial W^{(L)}} \tag{15}$$

$$\frac{\partial C}{\partial a^{(L)}} = 2\frac{1}{N}\sum_{n=1}^{N}(a^{(L)} - y_n) \tag{16}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma(z^{(L)})(1 - \sigma(z^{(L)})) \tag{17}$$

$$\frac{\partial z^{(L)}}{\partial W^{(L)}} = a^{(L-1)} \tag{18}$$

Backpropagation (II)

The derivate $\frac{\partial C}{\partial b^{(L)}}$ is similar,

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial W^{(L)}} \tag{19}$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1 \tag{20}$$

Backpropagation (III)

With $\frac{\partial C}{\partial W^{(L)}}$ and $\frac{\partial C}{\partial b^{(L)}}$, weight and bias in the first layer can be updated. Derivatives of the parameters in the previous layer can be found by chain rule.

$$\frac{\partial C}{\partial W^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}} \tag{21}$$

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = W^{(L)} \tag{22}$$

$$\frac{\partial z^{(L-1)}}{\partial W^{(L-1)}} = a^{(L-2)} \tag{23}$$

Single-Hidden-Layer NN

Homework

- Implement a backpropagation solver.
- Add it to an object function named *NeuralNetwork*
- Explore example with Boston dataset (regression).

References I

David Harris.
Digital design and computer architecture.
Elsevier/Morgan Kaufmann, Amsterdam Boston, 2012.

Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang.
Extreme learning machine for regression and multiclass classification.
IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(2):513–529, 2011.

References II

Alex Krizhevsky.
Learning Multiple Layers of Features from Tiny Images.
... Science Department, University of Toronto, Tech. ...,
2009.