

Introduction to computational models

Lab Assignment 1. Implementation of the multilayer perceptron

Javier Sánchez Monedero

`jsanchezm@uco.es`

Module “Introduction to computational models”

4th year of “Grado en Ingeniería Informática”

Especialidad Computación

Escuela Politécnica Superior

(Universidad de Córdoba)

20th September 2022



- 1 Contents
- 2 Notation and architecture
- 3 Data normalization
- 4 Pseudocode



Objectives of the lab assignment

- To familiarise the student with neural networks, in particular, with the multilayer perceptron.
- To implement the basic backpropagation algorithm for multilayer perceptrons.
- To check the effect of different parameters and data preprocessing:
 - Network architecture.
 - Learning rate.
 - Momentum factor.
 - Data normalization.
 - etc.



Backpropagation algorithm

- Please, read and analyse the theory notes.
- Pay special attention to the pseudo-code.



Stop condition

- **Standard version**, the algorithm stops if:
 - Training error does not decrease more than 0,00001 or increases, during 50 iterations (external loop).



Data normalization considerations I

- Normalization is an essential step in data pre-processing in any machine learning application and model fitting. This is essential to deal with the effect of different variable magnitudes and distribution of each variable.
- There are many normalization alternatives. Two widespread ones are:
 - Scaling: each feature is transformed within the range $[a, b]$:

$$X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}} \quad (1)$$

- Standardization: each feature is transformed so that the normalized feature has mean equal to 0.0 and standard deviation equal to 1.0:

$$X' = \frac{X - \mu}{\sigma} \quad (2)$$



Data normalization considerations II

- It is important to estimate the normalization parameters **using the training data**, and then to apply data transformation to the train and test sets with these parameters.
- A **typicall error** is to perform data normalization on the test dataset calculating minimum and maximum values ($X_{\text{máx}}$ and $X_{\text{mín}}$), or the mean and deviation (μ y σ), on the test dataset or before data splitting. In both cases we will be (wrongly) using testing information during our model building and calibration.
- For our work, we will implement feature scaling for input variables in $[-1, 1]$ and for output variables in $[0, 1]$ because we have a sigmoid transformation in the output layer.



Backpropagation algorithm

On-line backpropagation

Start

① $w_{ji}^h \leftarrow U[-1, 1]$ // Random values between -1 and $+1$

② Repeat

① For each pattern with inputs \mathbf{x} and outputs \mathbf{d}

① $\Delta w_{ji}^h \leftarrow 0$ // Changes will be applied for each pattern

② $out_j^0 \leftarrow x_j$ // Feed inputs

③ forwardPropagation() // Forward propagation ($\Rightarrow \Rightarrow$)

④ backPropagation() // Error backpropagation ($\Leftarrow \Leftarrow$)

⑤ accumulateChange() // Obtain the weight update

⑥ weightAdjustment() // Apply the calculated update

End for

Until (StopCondition)

③ Return weight matrices.

End



Backpropagation algorithm

Off-line backpropagation

Start

① $w_{ji}^h \leftarrow U[-1, 1]$ // Random values between -1 and $+1$

② Repeat

① $\Delta w_{ji}^h \leftarrow 0$ // Changes will be applied at the end

② For each pattern with inputs \mathbf{x} and outputs \mathbf{d}

① $out_j^0 \leftarrow x_j$ // Feed inputs

② forwardPropagation() // Forward propagation ($\Rightarrow \Rightarrow$)

③ backPropagation() // Error backpropagation ($\Leftarrow \Leftarrow$)

④ accumulateChange() // Obtain the weight update

End for

③ weightAdjustment() // Apply the calculated update

Until (StopCondition)

③ Return weight matrices.

End



Backpropagation algorithm: functions

forwardPropagation()

Start

- ① **For** h from 1 to H *// For each layer ($\Rightarrow \Rightarrow$)*
 - ① **For** j from 1 to n_h *// For each neuron of layer h*
 - ① $net_j^h \leftarrow w_{j0}^h + \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1}$
 - ② $out_j^h \leftarrow \frac{1}{1 + \exp(-net_j^h)}$

End For

End For

End



Backpropagation algorithm: functions

backPropagation()

Start

- ① **For** j from 1 to n_H *// For each output neuron*
 - ① $\delta_j^H \leftarrow -(d_j - out_j^H) \cdot g'(net_j^H)$ *// We have eliminated the constant (2), the result should be similar*

End For

- ② **For** h from $H - 1$ to 1 *// For each layer ($\Leftarrow \Leftarrow$)*
 - ① **For** j from 1 to n_h *// For each neuron in layer h*
 - ① $\delta_j^h \leftarrow (\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1} \delta_i^{h+1}) \cdot out_j^h \cdot (1 - out_j^h)$ *// Navigate all neurons in layer $h + 1$ connected with neuron j*

End For

End For

End



Backpropagation algorithm: functions

accumulateChange()

Start

- ① **For** h from 1 to H *// For each layer ($\Rightarrow \Rightarrow$)*
 - ① **For** j from 1 to n_h *// For each neuron of layer h*
 - ① **For** i from 1 to n_{h-1} *// For each neuron of layer $h - 1$*

$$\Delta w_{ji}^h \leftarrow \Delta w_{ji}^h + \delta_j^h \cdot out_i^{h-1}$$

End For
 - ② $\Delta w_{j0}^h \leftarrow \Delta w_{j0}^h + \delta_j^h \cdot 1$ *// Bias*

End For

End For

End



Backpropagation algorithm: functions

weightAdjustment()

Start

- ① **For** h from 1 to H // *For each layer ($\Rightarrow \Rightarrow$)*
 - ① **For** j from 1 to n_h // *For each neuron of layer h*
 - ① **For** i from 1 to n_{h-1} // *For each neuron of layer $h - 1$*

$$w_{ji}^h \leftarrow w_{ji}^h - \eta \Delta w_{ji}^h - \mu (\eta \Delta w_{ji}^h (t - 1))$$

End For
 - ② $w_{j0}^h \leftarrow w_{j0}^h - \eta \Delta w_{j0}^h - \mu (\eta \Delta w_{j0}^h (t - 1))$ // *Bias*

End For

End For

End



Introduction to computational models

Lab Assignment 1. Implementation of the multilayer perceptron

Javier Sánchez Monedero

`jsanchezm@uco.es`

Module “Introduction to computational models”

4th year of “Grado en Ingeniería Informática”

Especialidad Computación

Escuela Politécnica Superior

(Universidad de Córdoba)

20th September 2022

