## Neural and Evolutionary Computation (NEC)

# Activity 1: Prediction with Back-Propagation and Linear Regression

### Objective

The goal of this activity is to practice with different types of supervised models. In particular we are going to use the three following:

- **Multiple Linear Regression** (**MLR-F**), using the sklearn python library.
- **Neural Network with Back-Propagation** (**BP**), implemented by the student.
- **Neural Network with Back-Propagation (BP-F**), using a python library (pytorch/tensorflow).

### Deliverables

This assignment can be done alone or in pairs (groups of two)

For this activity you must deliver **one PDF document** that includes:

- A link to the Github repository where the code of all the activity is accessible. More details on the code in the following sections.
- Explanations of the analysis of the data preprocessing, and the results of the executions that are detailed in the section below.
- The name of the file should be **A1-Name1Surname1-Name2Surname2.pdf**

As an alternative, you can upload just a PDF with the link to the github (using the same name) and **include all the explanation and figures directly in the jupyter notebooks using markdown.**

### Part 1: Selecting and analyzing the dataset (5% of the grade)

The predictions must be performed on **one dataset** that the students can select. The dataset must have the following features:

- At least 10 input features and one output feature.
- The input features should include numerical AND categorical values.
- The prediction variable must take real (float or double) values; it should not represent a categorical value (that would correspond to a classification task)
- At least 1000 patterns
- Select randomly 80% of the patterns for training and validation, and the remaining 20% for test; it is important to shuffle the original data, to destroy any kind of sorting it could have.
- It is highly recommended that in case of doubt on the suitability of the selected data, **please ask the professor**.

As an output of this part, **you should include in your report the following information**:

- **Add to the documentation of this assignment the link to the source webpage.**
- Explain what techniques of data preprocessing you are going to apply: check for missing values, represent correctly categorical values, and look for outliers.
- Explain how you apply data normalization/transformation to some (or all) of the input / output variables.

Once this part is completed, you should generate the preprocessed / normalized files that are going to be the input of your analysis part.

**Part 2: Implementation of BP (40% of the grade)**

The second part of this activity is the programming of a neural network with back propagation from scratch. Here are some instructions on how to code the project.

- You should create a project in a github account where you are going to develop the activity. You should do regular commits to this project so we can observe the evolution of the project from the initial file to the final delivery. We will apply a penalization to the grade if there is one single commit in the project.

- We recommend the implementation of the project using Python (version >= 3.6). We are providing a base code that can be used to start the project (NeuralNet.py).

- In this part you must implement all the methods necessary for the network to learn, you cannot use external libraries that already implement BP or Neural Networks.

- The implementation must be based on the algorithm and equations in document [G] of Unit 3 at Moodle.

- The implementation must use the following variables to hold all the information about the structure of the multilayer neural network and the BP:
    - `L`: number of layers
    - `n`: an array with the number of units in each layer (including the input and output layers)
    - `h`: an array of arrays for the fields (h)
    - `xi`: an array of arrays for the activations (ξ)
    - `w`: an array of matrices for the weights (w)
    - `theta`: an array of arrays for the thresholds (θ)
    - `delta`: an array of arrays for the propagation of errors (Δ)
    - `d_w`: an array of matrices for the changes of the weights (δw)
    - `d_theta`: an array of arrays for the changes of the weights (δθ)
    - `d_w_prev`: an array of matrices for the previous changes of the weights, used for the momentum term ($\delta w^{(prev)}$)
    - `d_theta_prev`: an array of arrays for the previous changes of the thresholds, used for the momentum term ($\delta\theta^{(prev)}$)
    - `fact`: the name of the activation function that it will be used. It can be one of these four: sigmoid, relu, linear, tanh.

- For example, the weight $w_{ij}^{(L)}$ between unit `j` in layer `L-1` and unit `i` in layer `L` is accessed as `w[L][i,j]`

- The idea behind this structure is that the code must be able to deal with arbitrary multilayer networks. For example, a network with architecture 3:9:5:1 (4 layers, 3 input units, 1 output unit, and two hidden layers with 9 and 5 units, respectively), would have `n=[3; 9; 5; 1]`, and `xi` would be an array of length 4 (one component per layer), with `xi[1]` and array of real numbers of length 3, `xi[2]` and array of real numbers of length 9, `xi[3]` and array of real numbers of length 5, and `xi[4]` and array of real numbers of length 1. Similarly, `w[2]` would be an array `9x3`, `w[3]` an array `5x9`, and `w[4]` and array `1x5`; `w[1]` is not used.

- Additionally, the use of this structure, name conventions and array dimensions make it easy to convert the equations into code.

- The code will receive one input dataset, and using the percentage of data that is passed as a parameter in the class constructor, should divide this dataset into training and validation. If the percentage is 0, then we consider that there is no validation, and all the input data is used for training.

- The class NeuralNet **will receive all these parameters in the class constructor**:
    - Number of layers
    - Number of units in each layer
    - Number of epochs
    - Learning rate and momentum
    - The selected activation function (sigmoid, relu, linear, tanh)
    - The percentage of data that should be used as the validation set

- The class NeuralNet **will provide three public functions** that can be called externally:
    - A function **fit(X, y)** that has 2 parameters: an array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and a vector y of size (n_samples), which holds the target values (class labels) for the training samples. This method allows us to train the network with this data.
    - A function **predict(X)** that has 1 parameter, an array X of size (n_samples, n_features) that contains the samples. This method returns a vector with the predicted values for all the input samples.
    - A function **loss_epochs()** that returns 2 arrays of size (n_epochs, 2) that contain the evolution of the training error and the validation error for each of the epochs of the system, so this information can be plotted.

**Part 3: Obtaining and comparing predictions using the three models (BP, BP-F, MLR-F) – (35% of the grade)**

This part is focused on comparing the results of the backpropagation model that has been developed against two open-source implementations of machine learning methods.

The coding corresponding on this part should be done using jupyter notebooks, which should also be included in the github of the project.

To study the quality of the predictions in this section, we are going to use two elements:

1. The following evaluation metrics:
    - MSE - Mean Squared Error
    - MAE - Mean Absolute Error
    - MAPE - Mean Absolute Percentage Error

2. Visualizations of the results showing scatter plots of the prediction value $y^{\mu}$ compared with the real value $z^{\mu}$ for each of the models. The closer the points are to the diagonal, the better the prediction.

**Part 3.1: Hyperparameter comparison and selection**

First, we need to choose the correct set of hyperparameters that will provide the best prediction for the neural network that we have implemented. For this reason, we will have to estimate what are the optimal values for the network hyperparameters.

To find these parameters, must explore some of the space of hyperparameters, and evaluate the quality of the result of the prediction obtained using them. For this reason, you should include in the document **the following information**:

- A table that summarizes the quality of the prediction of each set of hyperparameters using the previously described metrics. You should try AT LEAST 10 combinations of parameters.

| Number of layers | Layer Structure | Num epochs | Learning Rate | Momentum | Activation function | MAPE | MAE | MSE |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

- 2 or 3 representative **scatter plots of the prediction value vs real value of some rows of this table** (including the set of parameters that gives the minimum the best metrics).

- 2 or 3 **plots of the evolution of the training and validation error as a function of the number of epochs** (the information that can be obtained calling (the function loss_epochs).

- A review/discussion of **2-3 paragraphs** of why you think these parameters are the most adequate.

**Part 3.2: Model result comparison**

Second, we are going to compare the results obtained in our network against two already implemented models. We are going to compare the best results obtained in the previous section against the following models:

- A multi-linear regression from scikit-learn (MLR-F).
- A neural network model, which can be used from Tensorflow, Pytorch, Scikit-learn or any other neural network python library (BP-F).

For this part **you should include in the document**, for each of the three datasets, the following information:

- The description of the parameters used in each of the two additional models.
- A table comparing the prediction quality for the three models.
- The scatter plot of predicted vs real values for the three models.
- 2 paragraphs of discussion comparing the results and functionality obtained by your implementation versus the other models.

**Optional Part 1: Study the effect of the different regularization techniques in the Neural Network (BP-F) (10%)**

In this part the goal is to introduce at least two regularization techniques to the neural network to try to improve the prediction results from those that we have seen in class (L1/L2 regularization, dropout, …).

For each of the techniques, you should:

- Explain how the regularization is introduced and the parameters used. It is important to justify why a certain value for a parameter is choosen, and it is recommended to try different values to observe the results.
- For each of the parameters, present the results of the prediction (evaluation metrics)

**Optional Part 2: Introduce Cross Validation in the model selection and validation (10%)**

The goal of this optional part is to improve your model evaluation process by incorporating cross-validation. Specifically, you are expected to implement k-fold cross-validation and use it to assess the performance of different parameter combinations.

Again, if you do this part you should include in the document a report that includes:

- How the cross-validation has been introduced, including the parameters (number of folds, and/or percentage of training patterns used, ..)
- Present the cross-validation results in the table format presented before, showing the mean and standard deviation of each performance metric (MSE, MAE, MAPE) for each model.
- Analyze the cross-validation results to determine which model generalizes best to unseen data.