# Project 1 – Neural Networks – Localization error estimation in WSN (Weeks 3.2-4.2)

## 1 – Introduction

Wireless Sensor Networks (WSN) are a main asset in the IoT. WSN are networks of spatially dispersed and dedicated sensors that monitor and record the physical conditions of the environment and forward the collected data to a central location. WSN can measure environmental conditions such as temperature, sound, pollution levels, humidity and wind, etc.

A WSN is built of "nodes" – from a few to hundreds or thousands, where each node is connected to other sensors. Each such node typically has several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting . A sensor node might vary in size from a shoebox to (theoretically) a grain of dust, although microscopic dimensions have yet to be realized. Sensor node cost is similarly variable, ranging from a few to hundreds of dollars, depending on node sophistication. Size and cost constraints constrain resources such as energy, memory, computational speed and communications bandwidth.

The **position information of sensor nodes** is obviously a very important issue and is widely needed in many areas such as routing, surveillance and monitoring. The node localization can be easily obtained with using Global Positioning System (GPS). However, this solution requires that all the sensor device nodes are equipped with GPS modules, which is a costly solution in terms of money and power consumption. The alternative is to equip only a few nodes with GPS (**anchor nodes**) and use them to calculate the position of the remaining nodes using connectivity information. Many range-free localization algorithms have successfully been proposed for this purpose. Such algorithms have low energy consumption, low cost, low noise sensitivity and high efficiency, but are usually time consuming. As such, performing many simulations using these methods, to decide how many sensors should be used to address a given problem, is often unfeasible in usable time. So, when one needs to decide how many sensors and how many anchor nodes should be used, or what should the transmission range be to guarantee an acceptable localization error, more time-efficient methods should be used.

To solve the described problem, an alternative solution has been proposed: perform experiments to obtain enough data to train a Neural Network that allows to quickly check what localization error is expected when one uses a given set of parameters. The resulting dataset has now been made available (Lab6-Proj1_Dataset.csv).

## 2 – Dataset

The "Lab6-Proj1_Dataset.csv" file is composed of 285 records. Each record contains the following fields:

- Node Density (number of sensors per $m^2$)
- Anchor Ratio (number of anchors per number of total nodes)
- Transmission Range (m)
- Step Size (parameter in the algorithm used to estimate sensor localization)
- Iterations (number of iterations in the algorithm used to estimate sensor localization)
- Estimated Sensor Location Error – ELSE (m)

Note that the dataset has not been preprocessed – it might contain noise, outliers, missing inputs, redundant features, etc.

## 3 – Objective

The objective of this project is to develop a NN model that can estimate the Sensor Localization Error (ESLE) in meters when given (some or all) the features contained in the collected dataset.

### 3.1 – Submission details and Deadline

This Project accounts for 40% of the Lab final grade. The final code and a comprehensive report must be submitted via Fenix until **Monday, May 29th, at 23:59**.

The students must submit:

a) All the developed code used to train and create the model.
b) A piece of code that will allow me to test your model using unseen data. This code, called TestMe, accepts as a parameter the "Lab6-Proj1_TestSet.csv" file that has the same structure of "Lab6-Proj1_Dataset.csv". The code must test the model you created on this new data (the program cannot "fit" the model to this new data, only "predict" it). The output of the code is:
    a. A vector containing the ELSE prediction for each of the inputs of the test file
    b. The RMSE of the ELSE prediction.
c) A report where you indicate the options you made regarding the data preparation, the experimental setup, the construction of the model(s), the evaluation and validation process, and the results you obtained. I will be especially picky with the experimental setup part. It's more important than obtaining an "exceptional" model. So, remember to take note of all decisions you make while checking and preparing the data, deciding the hyperparameters, etc., since it will be useful for the report.

## 4 – Implementation, Evaluation and Validation

Use all the knowledge you have acquired so far regarding data preparation, experimental setup and Neural Networks, to see how well you can solve this problem and build a good model (please re-check the theory slides regarding data and experimental setup before starting the project).

It is up to you to analyze the problem, look at the data, clean it, see which features might or might not be helpful (you can obviously try to use them all), check if new features might be useful, decide if the order is relevant, etc.

Note that this is a "regression" problem, not a "classification" problem.

Regarding the Neural Network, it's up to you to decide (or simply try) different architectures and the respective hyperparameters. I advise you to start with a "standard" MLP, using the most usual activation function (logistic), the most usual solver (stochastic gradient descent – sgd), not using regularization, etc. Note that these ARE NOT the default parameters for NN in Scikit, so you must indicate them explicitly.

Try to avoid OVERFITTING, since later I will check how your system performs under previously unseen data. The best way to know if there is overfitting or not is to use a Train/Validation/Test split, and only use the Test set once your model is completely defined (input features, number of layers, number of neurons per layer, activation functions, etc.). I.E, use a Holdout set! If after checking the results on the Test Set, you change any parameter or hyperparameter (even if it's a simple change in the number of neurons in a hidden layer), then you are accidentally fitting the model to the Test set (which might result in overfit).