->1 - Data pre-processing:

In order to get the data from excel, we used a code made in python, the .py file is called data\_.

The goal is to predict transmission rate of covid-19, so, firstly, we identified the predictors, which are: Temperature and Humidity (2 predictors).

Secondly, for every value that was 2 standard deviations from the mean (this for the predictors) or it wasn’t data at all (f.e. “A” or “#”), we replaced it with the value from the day before that day.

Thirdly, organize the predictors (inputs) according to the predictand (output), this means that, for each input, calculate, the **correlation** **coefficient** (c.c.) between the input and the output, then **lag** the input by 1 (use the data from 1 day before to predict the day we are at) and calculate the c.c. again, do this a couple of times.

Repeat this whole process, but this time instead of starting with the data from that day, start with the **weighted** **average** of that day and the day before (the data from today’s day has more weight than the data from yesterday f.e. 0.7 and 0.3), keep doing this, adding 1 to the number of days that you calculate the weighted average with, save that number of days and the lag by “x” which gets a c.c. closer to 1.

Remove rows with missing values.

Finally, in order to train and use the algorithm, we had to standardize (using the following formula: and divide the data into 3 subsets: **training** **data**, **validation** **data** and **test** **data**; because the goal is to predict the transmission rate of covid-19, choosing the data from all summers and springs as training data (≈50% of all data), the data from all winters as validation data (≈25%) and the data from all autumns as test data (≈25%), avoids major seasonal differences in values, in the subsets. Calling the function get\_data\_from\_excel gets these 3 subsets into 1 array each.

-> 2 - Implementation of the MLP algorithm:

To train our ANN, with the **backpropagation algorithm**, before anything else, we had to choose an activation function, we chose the **sigmoid** () and initialize 4 matrixes containing the weights between inputs and hidden nodes (A), weights between hidden nodes and outputs (C), bias for the hidden nodes (B) and bias for the outputs (D).

Diagram

Description automatically generated

D

C

A

B

Dimensions of each matrix:

A -> number of hidden nodes X number of inputs;

B -> number of outputs X numbers of hidden nodes;

C -> number of hidden nodes X 1;

D -> number of outputs X 1.

Initialize these matrixes, in every position, with random values belonging to this interval: , n stands for the number of inputs.

Afterwards, make a **forward pass** through the network, using inputs from the training subset, computing weighted sums,, activations for every node and store the results in 2 separate arrays called hidden and outputs, I did this with the **feed\_forward** function. Then, make a **backwards pass** computing for each node j, using the expected output according to the inputs (the **train** function does this):

A picture containing lamp, stand

Description automatically generated

-> use this for the output node ( is the expected output)

-> use this for the hidden nodes ( is the transpose matrix of )

For sequential processing -> train function returns the 4 matrixes with the **updated weights** ( is the learning rate, 0.1 initialy):

-> is the array with the inputs

-> is the array called hidden (defined above)

This whole process is based on **gradient descent**, where the local minimum of a certain function is calculated, iteratively, by taking steps proportionally to the negative of the gradient.

Adding improvements:

* Momentum -> adds a momentum term to gradient descent (proved efficiency ahead).

, when updating the weights, becomes: , the same for B, C and D matrixes, ;

* Annealing -> basically changes the learning rate every epoch based on the fact, that if we keep changing the learning rate, the network won´t get stuck in a minimum or oscillate to hard, just like in physics the higher the temperature the more active the atoms are, and as the system cools the activity of those atoms decreases. It changes according to this formula: , p = end parameter: 0.01, q = start