**14. Analysis Air quality data with Neural Network**

**Contact**: Guus Velders ([g.j.m.velders@uu.nl](mailto:g.j.m.velders@uu.nl); BBL 6.12), Oct. 2022

**Data**: Same as for Random Forest Hand-in

In the hand -in project air quality data was analyzed using a random forest model. Another popular method that can be applied is a Neural Network.

*Diagram

Description automatically generated*

Neural network connecting input data (meteorological data) to output (air quality).

**Task**: Build a random forest model for an air quality component based on meteorological parameters (wind, temperature, etc.), train the model with observations from 2015 to 2017 and make a prediction with the model for 2018. Test the performance of the model for 2018 by comparing model predictions with the observations for 2018.

1. Define a research question and think of how you are going to address it.

2. Use the same data and model as you used for the hand-in, but now apply a neural network (see for code below).

3. Analyze the results:

* Study the effect of the number of hidden layers and hidden nodes on the performance of the model
* Study the impact of the number of epochs and the number of batches on the performance of the model
* Compare the results with those from a random forest model (apply various statistics).

References:

* See the following website for useful info on programming a neural network in Python:

<https://machinelearningmastery.com/neural-network-models-for-combined-classification-and-regression/>

Code for Neural Network (similar to Random Forest)

🡺 First install tensorflow and keras

#=====================================================

# x = input

# y = output

# Import libraries

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the Multilayer Perceptron (MLP) neural network model with two

# hidden layers with 20 and 10 nodes

model = Sequential()

model.add(Dense(20, input\_dim=x\_train.shape[1], activation='relu',

kernel\_initializer='he\_normal'))

model.add(Dense(10, activation='relu', kernel\_initializer='he\_normal'))

model.add(Dense(1, activation='linear'))

print(model.summary())

# Compile the keras model: trained to minimize the mean squared error (MSE)

# loss function using the effective Adam version of stochastic gradient descent

model.compile(loss='mse', optimizer='adam')

# Fit the MLP keras model on the dataset

# Epochs: Number of epochs to train the model. An epoch is an iteration over

# the entire x and y data provided. The model is not trained for a

# number of iterations given by epochs, but merely until the epoch of

# index epochs is reached.

# Batch size: Number of samples per gradient update

model.fit(x\_train, y\_train, epochs=150, batch\_size=32, verbose=2)

# Make a prediction

y\_pred = model.predict(x\_pred)

#=====================================================