The university Albstadt-Sigmaringen is using two neuronal networks to detect the current state of the learning factory. To make this possible it is necessary to preprocess all data in a way that you can apply a neuronal network onto it. How to preprocess the data in that manner is described in the following chapter.

First of all, the data is acquired by receiving messages from each topic, which are the different stations of the learning factory. Each message comes with several information, such as the time stamp, the station, an acknowledge code, an active or inactive state and an optional description and target. To get a set of data to work with the ordering process is conducted once and the raw data is written into a JSON file and saved. Therefore, this data can be used in all of the following steps and it is not necessary to repeat an ordering process more often. A sample message from the high-bay warehouse is shown in the following figure.

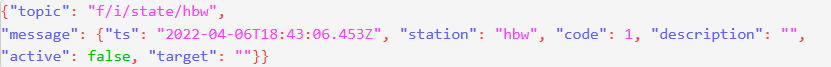


Figure x.x: Sample message

The raw data is not sorted correctly corresponding to the time steps, so the first step for data preparation is to run a simple sorting algorithm. The reason for the data to be in the wrong order is that the distance between some recorded time stamps is less than a millisecond and during transmission they can get mixed up. Data is read from the JSON file and written into a list, then the time stamps is extracted from the messages and a list with two columns is created, where the time stamps are in the first column and the message in the second. Now the list is sorted alphabetically by the time stamps and the messages can be written in a new JSON file. An example is given in the figure below.

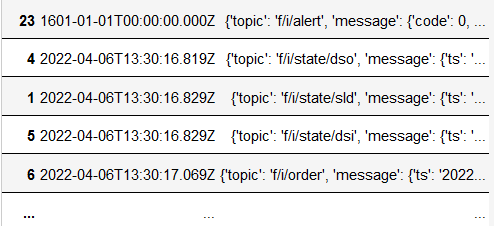


Figure x.x: Sorted messages

Now the raw data needs to be converted into a form that allows the models of the neuronal networks to work with it. The relevant data are the code, the active stand and the target, which needs to be values from the same data type and range in order for the model to work efficient. Since the codes are either 0, 1 or 2 and the active state is either 0 or 1 it makes sense to also convert the target to small integer values. In this case 0 to 3 are used to represent the different targets. With that done there are only 13 input data with small integer values left. The starting values of each input is set to -1 and updated whenever a new message is received, then the integer values are written into a new file. The initial state is reached as soon as there is a message from each station.

Table x.x: Possible active state, code and target of each station

|  |  |  |  |
| --- | --- | --- | --- |
| station | active | code | target |
| vacuum gripper robot | 0, 1 | 1, 2 | 0, 1, 2, 3 |
| sorting line with color detection | 0, 1 | 1, 2 |  |
| multi-processing station with oven | 0, 1 | 1, 2 |  |
| high-bay warehouse | 0, 1 | 1, 2 |  |
| input station | 0, 1 | 0, 1 |  |
| output station | 0, 1 | 0, 1 |  |

The data is provided in the following order:

hbwstate, hbwcode, vgrstate, vgrcode, vgrtarget, mpostate, mpocode, sldstate, sldcode, dsistate, dsicode, dsistate, dsocode

Next, every state that is possible during an ordering process has to be determined. This is achieved by sorting out all identical states, but only the consecutive ones, as well as all states that have a -1 in it. Then the remaining states have to be labeled manually, because otherwise the model would not now what they mean. For example the initial state (0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1) is found twice in that list because it occurs at the beginning as well as at the end of a ordering process.

If two or more states are represented by the same data, they have to be summarized under one name. This happened twice in the previous labeling so it has to be changed to a meaningful common name. Now there are 9 states left, which are easy to process in the model. Lastly the states also need to be assigned to corresponding integer values.

When all the data preparation and preprocessing is done, they can finally be loaded into the model of the neuronal networks. At first the convolutional neuronal network (CNN) is used and then a recurrent neuronal network (RNN) is applied.

For the CNN the data has to be separated into input and output data. As input the 13 values of the stations are used and for output the 9 states, which are put in a 1D-array.

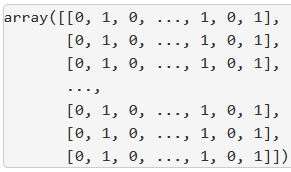
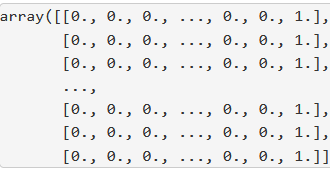


Figure x.x: Input array (left) and output array (right)

The following step is to set the definitions for the sequential model of the convolutional neuronal network. The model consists of 13 input neurons and 1 dias neuron that fires random values to ensure the model will not specify to much. A hidden layer with 10 neurons is added to ne model as well as the 9 output neurons and the activation functions are defined. When the model is fully defined it has to be configured using a compile function, where the standard optimizer is chosen.

When training the model with the input and output data, the results shows that it gets better and better for each epoch. After 100 epochs the accuracy is >99% and the mean absolute error is <2%. After 200 epochs the accuracy reaches 100% and the mean absolute error is near 0% and from then on the improvements are neglectable. When the model is now tested with sample input data is it expected that it always gives the correct result.

For the RNN the previous states are also put into consideration, leading to a faster and more precise result. The output data is the same as for the CNN put the input data has to be processed further as a 1D-array is not enough for the RNN. A multidimensional array with as many the previous states as wanted is created.

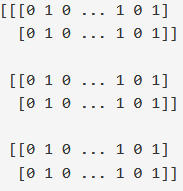


Figure x.x: Multidimensional input array

For model definition a different input layer called long short-term memory is required, which contains the 13 neurons, the dias neuron plus the previous states.

The model training follows the same procedure as for the CNN, but the results are worse this time, because after 500 epochs the accuracy is about 98% and the mean absolute error 2%. It seems like the RNN does not suit the application as well as the CNN does. When tested with the same sample data the results are also correct which indicates that the RNN is still good, but there are some issues with transitions between the output states.