Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 54-7 (2021) 535–540

Soft Sensor Transferability between Lines of a Sulfur Recovery Unit

F. Curreri * **L. Patanè** ** **M.G. Xibilia** **

* *Dipartimento di Matematica e Informatica, University of Palermo, via Archirafi 34, Palermo, 90123, Italy (e-mail: fcurreri@unime.it).*
** *Dipartimento di Ingegneria, University of Messina, Contrada di Dio, S. Agata, 98166 Messina ME, Italy (e-mails: lpatane@unime.it, mxibilia@unime.it)*

Abstract: Soft Sensors (SSs) are mathematical models that allow real-time estimation of hard-to-measure variables as a function of easy-to-measure ones in an industrial process, emulating the behavior of existing sensors when they are, for instance, taken off for maintenance. The Sulfur Recovery Unit (SRU) from a refinery is taken in exam. Recurrent Neural Networks (RNN) can capture the nonlinearity of such process but present a high complexity training and a very time-consuming structure optimization. For this reason, strategies to use pre-existing models are here examined by testing the transferability of the SSs between two parallel lines of the process.

Copyright © 2021 The Authors. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0)

Keywords: transferable soft sensor, nonlinear model, recurrent neural network, monitoring, prediction, inferential model

1. INTRODUCTION

Soft Sensors (SSs), also known as Virtual Sensors or Inferential Models (Graziani and Xibilia, 2020), represent a software model by which several measurement signals of a system are processed together to estimate the value of another variable of the same system. In an industrial environment, variables are monitored through online sensors. In some cases, some variables might be inaccessible or difficult to reach, or their hardware sensors work in a hostile environment with major disadvantages in terms of costs and maintenance. Sometimes they can even only be measured with high delays because of slow sensors or laboratory analysis. Such variables are then hard to monitor. SSs allow for real-time estimation of these hard-to-measure variables as a function of available data obtained from online sensors of the other easy-to-measure variables.

The use of these models leads to savings in terms of both immediate and prolonged spending. SSs, once designed, can continue to provide data and measurements for a very long time at no cost. They allow the development of tight control policies and they are used to approach several other different problems as well, such as measuring system back-up, what-if analysis, sensor validation and fault diagnosis (Kadlec et al., 2009).

As dynamic I/O models, they can be built with different approaches. Simpler and linear models employ the first principle modeling, based on physical knowledge of the system, in a white-box manner. But because of the complexity of industrial processes and the amount of available data, nonlinear models and data-driven black-box approaches are needed. Ad-hoc experiments should be performed to collect suitable datasets for the scope. But in practice, it is not always possible and data have to be retrieved from industries historical databases, generally provided by a supervisory control and data acquisition control system.

The design steps of an SS can be then summarized as follows (Souza et al., 2016):

(1) Data collection and filtering;
(2) Input variables selection;
(3) Model structure choice;
(4) Model identification;
(5) Model validation.

Once data have been collected and pre-processed to minimize oversampling, outliers and missing values, the best inputs representing the output(s) of the process in exam are chosen. Different techniques exist in the state of the art (Curreri et al., 2020a).

In the successive step, the model is chosen and the relative hyperparameters are selected. Different machine learning techniques have been used for data-driven designed SSs. The most employed are Artificial Neural Networks (ANN) and their variants, Deep Neural Networks, Support Vector Regression, Partial Least Square, Gaussian Processes Regression or Extreme Learning Machines, just to mention a few, as shown in works such as Fortuna et al. (2009), Curreri et al. (2020b), Sun and Ge (2019), Kaneko and Funatsu (2014), Shao and Tian (2015), Grbić et al. (2013), Shao et al. (2019).

Hyperparameters directly control the behaviour of the training algorithm and greatly impact the performance of the final model. Their optimization then plays a crucial role in the success of the SS. In the case of ANNs, they are represented by the number of hidden units and of hidden layers, the type of activation function of the neurons, the number of epochs needed for the training, or other

2405-8963 Copyright © 2021 The Authors. This is an open access article under the CC BY-NC-ND license.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2021.08.415

types of parameters specific to the chosen model. Once the structure is chosen and its hyperparameters optimized, the identification and validation steps are performed. Being data-driven models, SSs perform well if the distribution of training data and test data are the same. In the identification step, the first is used to empirically estimate the unknown parameters of the chosen model; the latter is employed, in the validation step, to verify whether the model can adequately represent the system and perform generalization to new never-seen-before samples. For this reason, models trained with data of one plant can not be employed to perform prediction for another plant. As shown in the work of Yang et al. (2019), even data collected from a plant can be slightly different from the ones collected from others of a similar type.

In this paper, a Sulfur Recovery Unit (SRU) of a refinery located in Sicily (Italy) is taken in exam. It is a dynamic process consisting of different lines that work independently, in parallel and with the same workflow. As discussed in the literature, it is a highly nonlinear process with clear dynamic dependencies between variables (Bolf et al., 2009). For this reason, among the available solutions adopted to build SSs, a Recurrent Neural Network (RNN) model was chosen in this work. Thanks to their ability to employ internal states (memory) to process the output(s), such networks are indeed more suitable to model dynamic time-dependent systems.

RNNs present anyway a high complexity training, as well as a high number of hyperparameters to be optimized. Being their training a very time-consuming task as well, strategies to re-use pre-existing models are here examined.

A transferable SS would play a key role in reducing the efforts needed to design an SS by adapting another one trained on a very close process. Transfer Learning (TL) algorithms focus on storing knowledge gained while learning a task and utilizing it for a different but related problem. TL methods adopted in Machine Learning are shown in the work of Weiss et al. (2016). Only recently, TL methods have been taken into account to design improved SSs. In the work of Farahani et al. (2020), a TL-based regression method is developed to design transferable SSs: in particular one able to adapt to a different plant of the same type, and another one able to work between different working conditions of the same process.

Similarly, this paper aims to test the transferability of an SS between two lines of the SRU. More specifically, two different SSs are firstly built, one for each line, with two different ad-hoc optimized structures. The models are then both tested on the other line, before being retrained. Finally, the structure of each model is used to create a new one for the other line, to test the transferability of the hyperparameters optimization.

The novelty of the proposed work consists in analyzing, through a series of steps, the trade-off between the accuracy and computational time in transferring an SS optimized for a specific process, to another closely related one. The role of hyperparameters and the effect on the model accuracy has been also investigated.

The paper is organized as follows: the process in exam is described in Section 2; RNNs are discussed in Section 3, along with their hyperparameters to be optimized. Results are shown in Section 4 and conclusions are finally drawn.

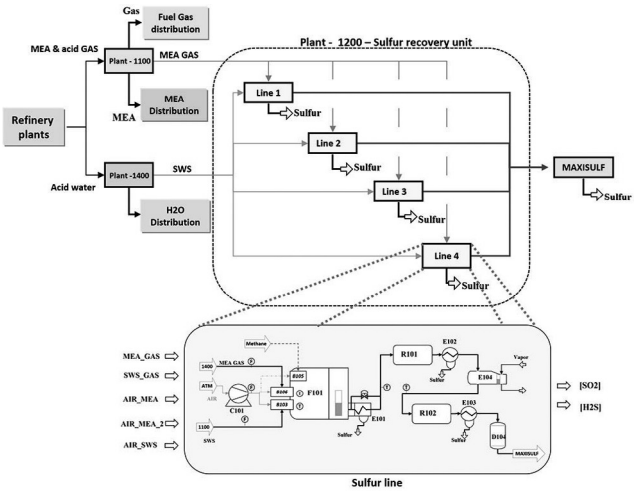

Fig. 1. Block diagram of the considered SRU that incudes four processing lines, here described through a simplified working scheme.

2. SULFUR RECOVERY UNIT

The role of SRU in refineries is fundamental both for the economic and environmental implications that this process involves. It exploits a gas desulfurizing process (or Clauss process) to recover elemental sulfur from gaseous hydrogen sulfide (H_2S), usually contained in by-product gases derived from refining crude oil and other industrial processes. This allows to both remove pollution from acid gas streams before they are released to the environment and recover sulfur as a valuable by-product as well. In particular, H_2S is a broad-spectrum poison for the human body: its effect prevents cellular respiration and it can affect several systems, mostly the nervous one.

The SRU studied in this work consists of four identical sub-units, called sulfur lines. They work in parallel, in the same way, transforming acid gases into sulfur. They have two types of acid gases as inputs: the MEA gas, a H_2S rich gas coming from gas washing plants; and another one coming from the Sour Water Stripping plant, indeed called SWS gas and rich in H_2S and ammonia (NH_3). A line is made of a furnace that has two separate combustion chambers. One is fed with MEA gas, whose combustion is regulated by an air flow supply (AIR_MEA). The other one is fed with SWS gas and its air flow (AIR_SWS), plus more MEA gas (MEA_SPILLING) and consequently more air flow (MEA_SPILLING_AIR) to keep the gas input flow constant. In the end, each line produces the same final gas stream, or tail gas, that contains a residual of H_2S and sulfur dioxide (SO_2). Moreover, a further air flow (AIR_MEA_2) is controlled by a closed-loop algorithm to improve the final stoichiometric ratio. The stoichiometric ratio $[H_2S] - 2[SO_2]$ is used indeed as feedback control for the air-feed ratio. Its desired value is zero, which implies that the pollutants are absent from the final product. The block scheme of the SRU process and the working scheme of an SRU line is shown in Fig. 1.

Online analyzers are used to measure the concentrations of both H_2S and SO_2 to compute the stoichiometric ratio. The sensor used for this measurement is often affected by reliability problems though since such components often cause damage in the online analyzer. When the latter suffers a break or a malfunction, due to the non-linearity of the system, nonlinear techniques are fundamental to generate a model that allows a reliable prediction of such ratio.

The SSs adopted in this process are used to simulate the concentrations of H_2S and SO_2 in the tail gas if a malfunction occurs. In this paper, two SSs were developed for line 2 and line 4. Even if sulfur lines work independently and in the same way, gas concentrations may vary both in input and in output between them.

To create the models, data have been collected from the historical database of the plant and the chosen inputs are the following:

(1) MEA_GAS: gas fed to the first chamber
(2) AIR_MEA: air fed to the first chamber
(3) AIR_MEA 2: further final air flow controlled by an algorithm
(4) SWS_GAS + MEA_SPILLING: total gas fed to the second chamber
(5) AIR_SWS + MEA_SPILLING_AIR: total air fed to the second chamber

3. RECURRENT NEURAL NETWORKS

The industrial process here analyzed, as shown in the literature (Fortuna et al., 2003), is highly nonlinear with clear dynamical dependencies between variables. In this work, among the available solutions adopted to develop an SS, an RNN was therefore selected to predict the concentrations of H_2S and SO_2. It is a dynamic model with a high-dimensional internal state that preserves some information about the input and output history. Therefore, it is not needed to explicitly feed delayed inputs into the network. RNNs (Rumelhart et al., 1986) from Feedforward Neural Networks, with which they share the structure: they are composed of a series of layers with a variable number of neurons and each neuron has directed connections to the ones of the subsequent layer through weighted arcs. Each neuron has its inputs summed up together to its bias. The difference lies in how the connection is formed since in an RNN connections between neurons are included between the previous levels and/or towards the same level, and not only in the direction of the next level. This makes this kind of networks able to show temporal dynamics behavior. In such a way, the output is influenced by both the current time instant and the inputs from previous time instants.Their ability to use their internal states makes them more capable to accomplish more complex types of tasks, such as handwriting recognition and speech recognition (Graves and Schmidhuber, 2009).

From a practical point of view, the connections between the output of a layer m and the input of the previous layer l are performed by applying a real-valued time-delay between them. This is done by Tapped Delay Lines (TDL), which contain delay operators z^{-d}, which delay time-discrete signals by a real-valued delay d. In Fig. 2

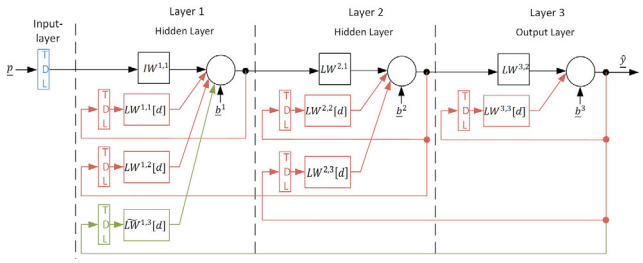

Fig. 2. RNN with two hidden layers with delays and recurrent connections.

an RNN with two hidden layers presenting delays and recurrent connections is shown, where the matrix $IW^{1,1}$ contains the connection weights of the first layer, while the matrices $LW^{l,m}$ contain the weights that connect the outputs of layer m with layer l. The value b^i represents the bias of the i-th neuron.

Three different types of TDL can be described:

- Input TDL (blue): Delays the inputs of the network by any real-valued time-step $d \geq 0$. This allows to model systems in which the output depends on current and previous inputs as well.
- Output TDL (green): Adds a recurrent connection of the outputs of the RNN to its first layer, for systems in which the outputs depend not only on the inputs but on previous states as well.
- Internal TDL (red): Adds a recurrent connection from all layers to all previous layers and to itself, except from the output layer to the first layer, for systems in which the output depends on previous internal states.

All other forward connections only have un-delayed direct connections.

Given the above definitions, every RNN presents the following hyperparameters to be optimized:

(1) Number of input delays
(2) Number of internal delays
(3) Number of output delays
(4) Number of hidden layers
(5) Number of neurons for each hidden layer
(6) Number of training epochs

The activation functions generally used are the hyperbolic tangent for all neurons in the hidden layers; the linear function for all neurons in the output layer. Moreover, two training algorithms have been taken into account: the Levenberg-Marquardt algorithm (Marquardt, 1963), and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Werbos, 1990).

Given the complexity of such networks, various difficulties have often been encountered mainly due to problems during training, known as gradient vanishing and exploding gradient. The first one occurs when the gradient becomes vanishingly small, at the point of preventing the weights from changing value; on the contrary, the second one occurs when the gradient increases exponentially making the derivatives eventually explode. Moreover, the gradual change of system hyperparameters often leads to bifurcations that are impossible to be read.

From a computational point of view, the presence of cycles involves a high cost, allowing one to work only with networks with a limited number of neurons.

4. RESULTS

Available data consist of 14401 samples from line 2 and 10081 from line 4, sampled every minute. Outliers presence was visually analyzed and they were manually removed by interpolation. Data were then normalized using z-score normalization. Being the system time-dependent, model training and testing were performed on time-continuous blocks of data. In particular, the first 80% of data was used to train the models and the remaining 20% to test them. All the models created have five inputs, enumerated in Section 2, and two outputs, respectively the concentrations of H_2S (output 1) and SO_2 (output 2). Therefore, a unique SS has been designed to predict both output variables in the selected lines. The following simulations have been performed in the Matlab environment using dedicated toolboxes and the computer configuration is as follows: OS: Windows 10 (64bit); RAM: 8GB; CPU: Core i5-9300H (2.4 GHz); Matlab version: 2020a.

4.1 Best models

The best performing models for both lines are here presented. Their hyperparameters optimization was carried with a double grid search. The first was employed to find the best combination of input, internal and output delays, leaving the architecture fixed, evaluating the final performance of the models in terms of Pearson Correlation Coefficient (CC) and Mean Squared Error (MSE) between the network outputs and the targets on test data. Once the best performing delays were found, a grid search was carried for the internal architecture of the network in the same fashion. Fig. 3 shows the relation between the network performance on test data and the network structure in terms of number of hidden layers and neurons, only for output 2 of SRU line 2. The final chosen architecture was eventually evaluated based on the performances of both outputs. Such best performing structure was then investigated with both LM and BFGS training algorithms. Such analysis was carried for both lines, leading to two very different final structures.

Optimized hyperparameters are listed in table 1: adopted delays, hidden layers structure, used training algorithm and the number of training epochs is shown. The computational time requested to optimize the network hyperparameters is considerable high. In the first grid search, delays of the inputs, internal states and outputs were considered from 1 to 5 steps, leading to 125 possible combinations. For each structure, five networks were considered and initialized with random weights, to carry a statistical analysis. This required about 100 hours of training in computational time. The internal architecture search was carried from 1 to 3 hidden layers of the same number of neurons. The latter was varied between 2 and 5. Further increasing the number of hidden layers and neurons led to extremely poor-performing models and to a higher probability of exploding gradient during training. For each of the possible architectures mentioned above, five networks, each one with a different weight initialization, were created and trained. This led to a total training time of more than 10 hours.

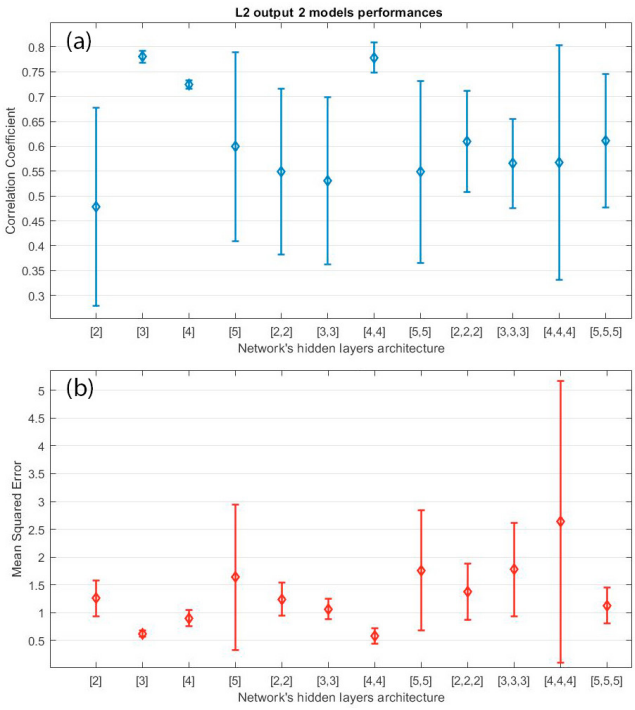

Fig. 3. Statistical distribution of the CC (a) and MSE (b) on test data for different network architectures (i.e., from 1 to 3 hidden layers with different numbers of neurons), in relation to the output 2 of line 2 model. The circles indicate the mean value whereas the bars represent the standard deviation.

Time could be reduced by applying more sophisticated searching strategies based on genetics algorithms, Bayesian Optimization, Tree-structured Parzen estimators (Bergstra et al., 2011; Falkner et al., 2018; Franceschi et al., 2017) but, in any case, it will represent the most time-consuming part of the learning process.

The time needed to train the optimal model, after the hyperparameters selection, for line 2 was about 29 minutes whereas for line 4 it was about 18 minutes. The time differences are attributable to the dataset size and to the different complexity of the two models as reported in table 1.

Table 1. Best models hyperparameters.

	Delays	Hidden layers	Train. alg.	Epochs
Line 2	Input: 1 Internal: 2 Output: 2	[4,4]	BFGS	300
Line 4	Input: 4 Internal: 5 Output: 1	[3,3]	BFGS	300

Table 2 shows the final performances in term of CC and MSE on test data. In Fig. 4 the networks outputs and the actual targets are shown for output 2 of both line 2 and line 4.

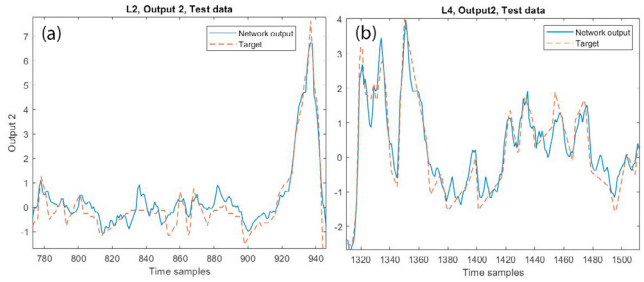

Fig. 4. Networks outputs and targets comparison of the output 2 of line 2 (a) and line 4 (b) best models on test data.

4.2 Transferred models

The two optimized models were tested each one on the other line to evaluate their performance on a different but similar process. As expected, the performances dropped, but the general trend of the target is still followed. These performances are listed in table 3. Fig. 5(a) shows the network output 2 of line 4 of the transferred model. It can be noticed that the possibility to use the SS optimized for one line directly into the other is still able to provide indications on the behaviour of the outputs, even though the performances are considerably lower when compared to the optimal performance obtained (reported in table 2). The significant advantage of this solution consists in the reduction of the time needed to develop one of the SS reducing the computational time by half.

4.3 Fine-tuned transferred models

To improve the performances of the transferred models, the two networks were fine-tuned on the new lines. The line 4 model was fine-tuned on the line 2 data with the LM algorithm with 6 epochs, starting from the previously learned weights. Such retraining took only 45 seconds. The line 2 model was fine-tuned on the line 4 data with the LM algorithm and 20 epochs: this took 2.5 minutes. These were the maximum number of epochs possible to avoid overlearning. The new performances are listed in table 4. Fig. 5 (b) shows how the network output 2 of line 4, compared to the target, improves after the fine-tuning.

4.4 Hyperparameters transferability

Finally, two new models were learned for both lines 2 and 4, adopting the structure of the other one, to test the

Table 2. Performances obtained optimizing the RNN for each line.

	Output 1	Output 2
Line 2	CC=0.84 MSE= 0.42	CC=0.84 MSE= 0.38
Line 4	CC=0.90 MSE= 0.23	CC=0.91 MSE= 0.20

Table 3. Transferred models performances

	Output 1	Output 2
Line 2	CC=0.65 MSE= 2.24	CC=0.70 MSE= 1.37
Line 4	CC=0.47 MSE= 1.03	CC=0.55 MSE= 0.82

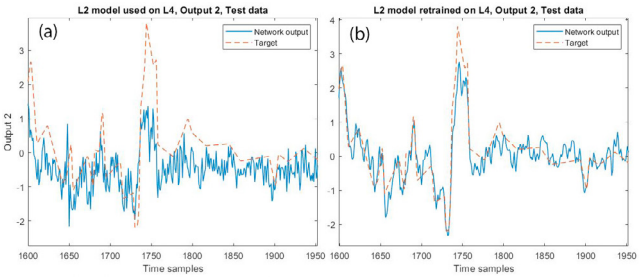

Fig. 5. Network output 2 and target comparison of the transferred model on line 4 before the fine-tuning (a) and after (b).

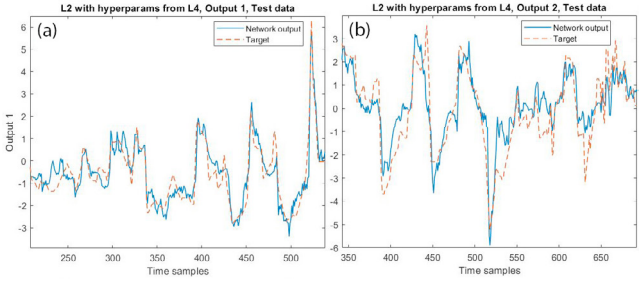

Fig. 6. Network output 1 (a) and 2 (b) compared to the targets for line 2 in the new model built using the hyperparameters originally optimized for line 4.

transferability of the hyperparameters, but starting from a random initialization of the network weights. The model for line 2 was trained with the BFGS algorithm in 200 epochs: the training took almost 16 minutes. The model for line 4 was trained with the BFGS algorithm in 327 epochs, taking almost 33 minutes. Final performances, the adopted training algorithm and the number of training epochs are listed in table 5. Fig. 6 shows the network outputs 1 and 2 compared to the targets of the new model for line 2.

5. CONCLUSIONS

The high nonlinearity characterizing processes like the ones that occur in an SRU, require a model able to capture the dynamic time-dependencies between variables. Even if RNNs are able to fulfil this requirement, the creation of such networks is a very demanding task. RNNs show various hyperparameters in their structure to be optimized. Moreover, their training time requires a high

Table 4. Fine-tuned transferred models performances

	Output 1	Output 2	Train. alg.	Epochs
Line 2	CC=0.75 MSE= 0.66	CC=0.78 MSE= 0.58	LM	6
Line 4	CC=0.80 MSE= 0.43	CC=0.86 MSE= 0.27	LM	20

Table 5. New models with transferred hyperparameters performances

	Output 1	Output 2	Train. alg.	Epochs
Line 2	CC=0.74 MSE= 0.58	CC=0.73 MSE= 0.93	BFGS	200
Line 4	CC=0.85 MSE= 0.31	CC=0.90 MSE= 0.2	BFGS	327

computational effort. For this reason, methods to use pre-existing models are needed. The transferability of SSs between lines 2 and 4 of the same SRU of a refinery plant located in Sicily was then examined.

The best performing models, along with their optimal structure, were first developed. Both the structure optimization and the training were very time-consuming task. Results showed good performances for both outputs and lines, although line 2 proved to be more difficult to model. The two SSs were then tested on the other line, showing a significant drop in performances. A fine-tuning on the new lines dataset showed a great improvement in performances with a limited effort in computational time since the re-training was performed for a reduced number of epochs without the need to optimize the other hyperparameters involved.

Finally, the optimal set of hyperparameters associated to each line was used to create a new model for the other one, with random initialization of the network weights. Such an approach would allow to skip the very time-consuming step of the structure optimization and led to acceptable performances, even though still not at the level of the best performance obtained optimizing the whole network structure.

Results showed that fine-tuning pre-existing networks of a similar process or adopting the same structure to create a new one are potential working solutions for fast analysis. The transferability of SSs is anyway a challenge that needs to be tested in the effort of finding new methodologies, even by investigating TL approaches in the SSs field.

REFERENCES

Bergstra, J., Bardenet, R., Kégl, B., and Bengio, Y. (2011). Algorithms for hyper-parameter optimization. *Conference: Advances in Neural Information Processing Systems*.

Bolf, N., Mohler, I., Golob, M., Galinec, G., and Duplančić, M. (2009). Software sensor for sulphur recovery unit control. *Chemical Engineering Transactions*, 17.

Curreri, F., Fiumara, G., and Xibilia, M.G. (2020a). Input selection methods for soft sensor design: A survey. *Future Internet*, 12.

Curreri, F., Graziani, S., and Xibilia, M.G. (2020b). Input selection methods for data-driven soft sensors design: Application to an industrial process. *Information Sciences*, 537, 1 – 17.

Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv:1807.01774*.

Farahani, H.S., Fatehi, A., Shoorehdeli, M.A., and Nadali, A. (2020). A novel method for designing transferable soft sensors and its application. *arXiv:2008.02186*.

Fortuna, L., Graziani, S., and Xibilia, M.G. (2009). Comparison of soft-sensor design methods for industrial plants using small data sets. *IEEE Transactions on Instrumentation and Measurement*, 58(8), 2444–2451.

Fortuna, L., Rizzo, A., Sinatra, M., and Xibilia, M. (2003). Soft analyzers for a sulfur recovery unit. *Control Engineering Practice*, 11(12), 1491 – 1500. Award winning applications-2002 IFAC World Congress.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization.

Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in Neural Information Processing Systems*, 21, 545–552.

Graziani, S. and Xibilia, M. (2020). Deep learning for soft sensor design. *Development and Analysis of Deep Learning Architectures, Springer*, 31–59.

Grbić, R., Slišković, D., and Kadlec, P. (2013). Adaptive soft sensor for online prediction and process monitoring based on a mixture of gaussian process models. *Computers & Chemical Engineering*, 58, 84 – 97.

Kadlec, P., Gabrys, B., and Strandt, S. (2009). Data-driven soft sensors in the process industry. *Computers & Chemical Engineering*, 33(4), 795 – 814.

Kaneko, H. and Funatsu, K. (2014). Adaptive soft sensor based on online support vector regression and bayesian ensemble learning for various states in chemical plants. *Chemometrics and Intelligent Laboratory Systems*, 137, 57 – 66.

Marquardt, D.W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2), 431–441.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.

Shao, W., Ge, Z., Song, Z., and Wang, K. (2019). Nonlinear industrial soft sensor development based on semi-supervised probabilistic mixture of extreme learning machines. *Control Engineering Practice*, 91, 104098.

Shao, W. and Tian, X. (2015). Adaptive soft sensor for quality prediction of chemical processes based on selective ensemble of local partial least squares models. *Chemical Engineering Research and Design*, 95, 113 – 132.

Souza, F.A., Araújo, R., and Mendes, J. (2016). Review of soft sensor methods for regression applications. *Chemometrics and Intelligent Laboratory Systems*, 152, 69 – 79.

Sun, Q. and Ge, Z. (2019). Probabilistic sequential network for deep learning of complex process data and soft sensor application. *IEEE Transactions on Industrial Informatics*, 15(5), 2700–2709.

Weiss, K., Khoshgoftaar, T.M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3, 9.

Werbos, P.J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), 1550–1560.

Yang, B., Lei, Y., Jia, F., and Xing, S. (2019). An intelligent fault diagnosis approach based on transfer learning from laboratory bearings to locomotive bearings. *Mechanical Systems and Signal Processing*, 122, 692 – 706.