Classification of PROFINET I/O Configurations utilizing Neural Networks

Bjarne Johansson[1,2], Björn Leander[1,2], Aida Čaušević[2], Alessandro V. Papadopoulos[2], Thomas Nolte[2]
[1] ABB Industrial Automation, Process Control Platform, Västerås, Sweden
[2] Mälardalen University, Västerås, Sweden
{bjarne.johansson, bjorn.leander}@se.abb.com,
{aida.causevic, alessandro.papadopoulos, thomas.nolte}@mdh.se

***Abstract*—In process automation installations, the I/O system connect the field devices to the process controller over a fieldbus, a reliable, real-time capable communication link with signal values cyclical being exchanged with a 10-100 millisecond rate. If a deviation from intended behaviour occurs, analyzing the potentially vast data recordings from the field can be a time consuming and cumbersome task for an engineer. For the engineer to be able to get a full understanding of the problem, knowledge of the used I/O configuration is required. In the problem report, the configuration description is sometimes missing. In such cases it is difficult to use the recorded data for analysis of the problem.**

In this paper we present our ongoing work towards using neural network models as assistance in the interpretation of an industrial fieldbus communication recording. To show the potential of such an approach we present an example using an industrial setup where fieldbus data is collected and classified. In this context we present an evaluation of the suitability of different neural net configurations and sizes for the problem at hand.

I. INTRODUCTION

The analysis of network data using Machine Learning (ML) techniques for, e.g., anomaly detection [1], or application classification [2], is an active research area [3]. However, in the domain of Industrial Automated Control Systems (IACS), especially within the Operation Technology (OT) network, it has not been widely explored, except of a few recent exceptions mostly found within the security domain, e.g., [4], [5]. During testing as well as after system commissioning, faults can occur in an I/O system. It is a common task for a support engineer to analyze recorded fieldbus data when troubleshooting such errors. When performing an analysis, it is essential to know the signal map, i.e., the signal value distribution in the recorded packets — needed for finding deviating signals in the recording. The I/O configuration provides this information. It is difficult to deduce information about the I/O configuration manually, due to the amount of recorded data needed to be recorded to understand the pattern.

In this work, we propose a solution that utilizes deep neural networks as an aid for determining the I/O configuration from a log containing recorded cyclic PROFINET I/O [6] data. PROFINET I/O is an Ethernet-based fieldbus and one of the most commonly used with a large installed base.

This work is funded by The Knowledge Foundation (KKS), project ARRAY, and by The Swedish Foundation for Strategic Research (SSF).

The contribution of this paper is twofold. First, we show that it is possible to use deep learning to recognize and classify I/O configurations by analyzing recorded PROFINET data. Second, to find a suitable model for our problem concerning neural network size, we empirically assess appropriate configurations and sizes. Additionally, we provide future direction for addressing analysis of communication recordings.

The paper is organized as follows. In Section II we describe the background and the system used in our work. Section III focuses on the test execution and presents the result, followed by Section IV in which we look at earlier, related work, and discuss our result. Finally, Section V, concludes the paper.

II. BACKGROUND

A. System description

Fig. 1 shows a schematic of the modular ABB Select I/O^{TM} [7] system. From a conceptual level, it is composed of (from bottom to top):

- I/O Modules, that connects directly to the process input and output signals. Currently 4 major types of I/O modules exist: Analog Input and Output modules (AI/AO) and Digital Input and Output modules (DI/DO).
- Redundant GIS880 modules summarizing data from connected I/O modules. Each redundant GIS880 module can at most serve 16 I/O modules.
- Redundant CI845 modules forwarding data from the GIS880 upward in the chain. Each CI845 can at most serve 12 GIS880 per Modulebus. A CI845 can therefore serve up to $3 \cdot 12 \cdot 16 = 576$ I/O modules at most.
- CI871 modules forwards data from CI845 to the logical controller unit. Each CI871 can serve a number of CI845s.

One can notice that the number of possible permutations of I/O configurations per CI845 units is large ($576^4 > 10^{11}$).

The I/O-modules communicate with the CI845 using the Modulebus communication bus. The CI845 transforms communication to the PROFINET format and forwards data to the CI871, which in turn is connected to the control system and the high level Distributed Control System (DCS). Modulebus is an ABB proprietary communication bus used to enable communication between the fieldbus condensing unit, in this

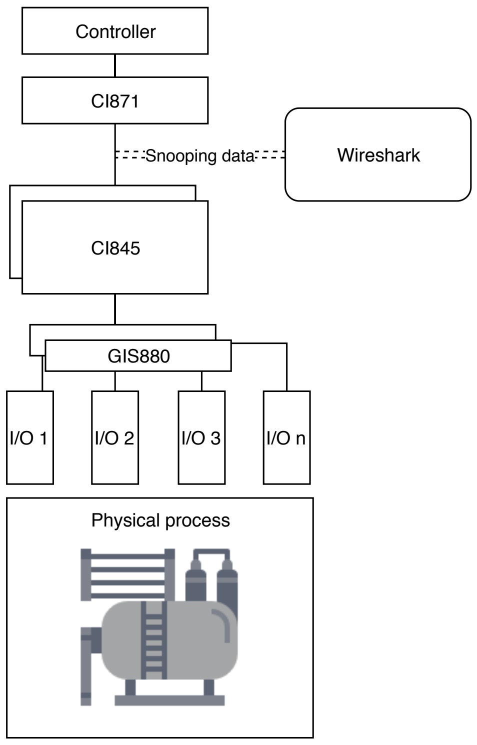

Fig. 1. System setup of the ABB Select I/O$^{\text{TM}}$.

case, the CI845, and the GIS880. PROFINET I/O is one of the leading Ethernet based fieldbus communication standards. PROFIsafe [8] is a standard related to PROFINET I/O tailored for safety-critical communication.

In this paper we use *Wireshark*[1] as a network protocol analyzer, to collect and analyze the traces of the system.

B. ML implementation techniques

In order to analyze the traces collected with Wireshark, we have implemented a neural network, implemented in Keras[2], a high-level neural network API that can be interfaced with different ML platforms, e.g., Tensorflow[3] [9].

Since the main objective of this work is to recognize and classify I/O configurations from the PROFINET data, we have designed a Multi-Layered Perceptron (MLP) [10] neural network, and used it as a classifier.

In the MLP, each node is a Threshold Logic Unit (TLU), and each node in layer ℓ is connected to all the nodes in the next layer $\ell + 1$, as shown in Fig. 2. Each node in the MLP has a set of n inputs $x_i \in \mathbb{R}$, each one with a given weight $w_i \in \mathbb{R}$. The output y of a node is computed based on the weighted sum of the inputs:

$$a = \sum_{i=1}^{n} x_i w_i \tag{1}$$

[1]https://www.wireshark.org/
[2]https://keras.io/
[3]https://www.tensorflow.org/

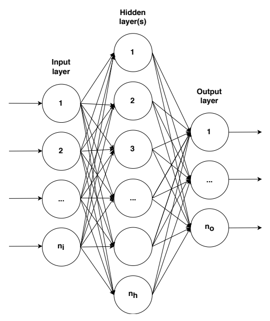

Fig. 2. An illustration of Multi-layered Perceptron.

and on an activation function so that:

$$y = f_{\text{act}}(a) \tag{2}$$

The vector of all the weights in the network $\mathbf{w}$ must be assigned with a value that depends on the application at hand, and this requires a training phase of the MLP, based on some labeled data. The weights $\mathbf{w}$ are updated by presenting learning data to the network, i.e., sets of input-values $\mathbf{v}$ combined with well known target values $\mathbf{t}$. The weights for the vectors are updated by applying a backpropagation algorithm. For one weight this is described as:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha(t - y)\mathbf{v} \tag{3}$$

where k refers to the k-th iteration of the backpropagation algorithm, t is the target output, y is the actual current output based on input values in vector $\mathbf{v}$. The constant α, is called the learning rate, and indicates how big the changes of the weights will be for each iteration.

In order to avoid the problem of overfitting, there is a method named *Dropout* [11], that can be applied to an input vector to a layer, during each pass of training:

$$\mathbf{v}' = Dropout(\rho, \mathbf{v}) \tag{4}$$

which will set a fraction $\rho, (0 < \rho < 1)$ of the values in $\mathbf{v}$ to zero. In this way an amount of noise is added to the relationship between layers and the training-data, reducing the risk of overfitting.

III. EXAMPLE/TEST SETUP

In this section, we describe the test environment along with the configuration classes. We also explain the implementation of the learning algorithms and we describe how we extract the learning and verification data.

A. Experimental setup

The I/O system used for the experiment is the ABB Select I/O$^{\text{TM}}$ described in Section II. Permutations of DO and AO modules in three positions is used for generating data. The high integrity version of the I/O is used since that utilizes PROFIsafe to realize a highly reliable communication channel. That also means that PROFIsafe containers are encapsulating the process values. The PROFIsafe container data changes cyclically even if the process values do not. The characteristics of the changes are, to some extent, module type dependent.

B. *Data extraction and partitioning*

Data extraction is performed using the Wireshark tool, configured to snoop traffic between the CI845 and CI871 devices (as shown in Fig. 1), using an industrial network switch allowing port mirroring and directing Wireshark to read data from the mirrored port. The data is then filtered such that only the cyclical data frames remain. The resulting data is saved in the K12 text-format. Classes with related dataset characteristics are summarized in Table I. The data-frames are normalized with respect to size such that each frame is zero-padded to consist of 112 bytes of data.

The recorded data is divided into three categories: i) training data (TrD), ii) test data (TeD) and iii) and prediction data, which is collected at a later point and serves as an example of real prediction data (RPD). At this later point, the process values are changed to contain values that are not used when training the model.

C. *Model description and evaluation*

Since the complexity of our data is relatively low, we choose to use a MLP neuron network with a layout as described in Table II. The Rectified Linear Unit ($ReLU$) activation function is used on the hidden layers. The ReLU method is used to forward only values greater than 0 to the second layer, i.e.,

$$ReLU(a) = \max(a, 0) \tag{5}$$

As this is a classification problem, the normalized exponential function (soft-max) is used for activation on the output layer [12].

$$S(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{6}$$

Between the two hidden layers, we add a dropout-method that mitigates the risk of overfitting [11]. This introduces an amount of uncertainty, and in the experiments performed it could be seen that the accuracy of the model would vary quite a lot between different executions.

The tuning of the model, e.g., the backpropagation and update of vector weights, is implemented using the Adam optimizer [13].

TABLE I
NUMBER OF SAMPLES PER DATASET AND CONFIGURATION

Class	Configuration	TrD	TeD	RPD
C1	DO	480	120	0
C2	AO	470	117	0
C3	AO DO	500	125	0
C4	DO AO	529	132	137
C5	AO AO	459	115	0
C6	DO DO	491	122	104
C7	AO AO AO	484	121	0
C8	AO AO DO	500	125	0
C9	AO DO AO	479	120	0
C10	DO AO AO	466	116	0

TABLE II
MODEL DESCRIPTION FOR LAYERS.

Layer	# TLUs	Activation
Input Layer	112	N/A
Hidden Layer 1	L_1	$ReLU$
Droupout(0.5)		
Hidden Layer 2	L_2	$ReLU$
Output Layer	10	Softmax

The cross-entropy loss-function is used to assess the relative accuracy of the results during the training phase, as is commonly used in classification problems [14]. The cross-entropy is calculated, for each class c the loss ℓ_c, as:

$$\ell_c = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{7}$$

where M is the number of classes, $y_{o,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for observation o and $p_{o,c}$ is the predicted probability that o is of class c.

In our experiments, we use ten epochs for the training phase. We use Keras default settings for weight initialization and learning rate, Glorot uniform initializer [15] as the weight initialization and a 0.001 learning rate. Empirically, the model converge within a few epochs, i.e., two to four, but in the experiments performed we did allow for a higher number of epochs to further refine the model, without significantly impacting the training time.

As for the design of the network size, we consider different sizes of the layers, (L_1 and L_2) using permutations of sizes in 2^n, $n \in \{4, \ldots, 12\}$, to find sizes that perform the best with our data with respect to accuracy. In the experiment, the size of the input layer is fixed at 112 neurons, as this is maximum payload in bytes. A variant would have been to use one neuron per bit. The output layer is also fixed at ten neurons since this is the number of classes. The following steps are performed for each iteration:

1) The model is trained with the training data (TrD).
2) The model is evaluated with the test data (TeD) and with the prediction data of the second recording (RPD).

D. *Results*

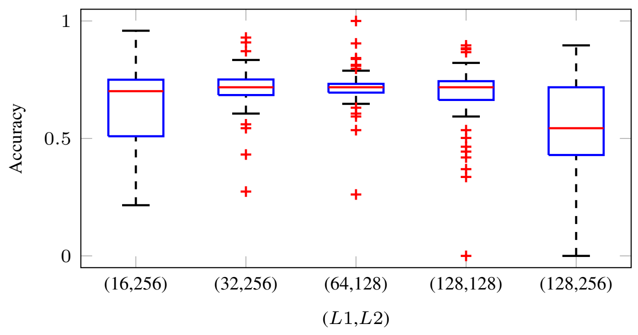

Fig. 3. Model accuracy variation distribution over 50 iterations on RPD data.

Fig. 3 shows the accuracy distribution for five different configurations of the neural network, over 50 iterations on the RPD data. The size of $L1$ and $L2$ are indicated as a tuple on the x-axis. For our data, models with a $L1$ size within the range $16 - 64$ neurons and $L2$ sizes within $32 - 1024$ can perform acceptable predictions. The accuracy for TeD for these sizes are in the range 80%-95%, while RPD accuracy are found between 60% and 80%. The model with the highest accuracy of prediction data is the most proper one for use in actual prediction scenarios. High accuracy is also reached for the RPD data, this accuracy is expected to be even higher if a broader range of signal values would be available in the training data.

One of the better performing configurations is the configuration where $L1$ consist of 64 and $L2$ consist of 128 neurons. This configuration has an accuracy on the TeD data over 90% and over 80% on the RPD data.

IV. Discussion

We are not the first to use ML for log and/or network communication recording analysis. Du et al. [16] developed DeepLog, a tool for anomaly detection in collected log data. DeepLog uses a Long Short Term Memory (LSTM), deep neural network model, to learn from log patterns and recognize system anomalies.

From our result, we can see that using ML for PROFINET I/O configuration prediction is possible. The accuracy is highly dependent on the amount and quality of the data used for training. Fig. 3 shows that the best performing network configuration can still have some outliers for the accuracy as low as 30%. Also, the quality of a trained network with the suggested model varies hugely. The reason for this nondeterministic behavior remains to be investigated.

The number of classes that are used for classification is the biggest weakness in this work. The goal of the work is to be able to deduce the I/O configuration from a recorded PROFINET I/O data log containing process data frames that are cyclically exchanged. Using the suggested approach we can reliably classify a tiny fraction of those configurations. Expanding the neural network and training data to span all possible arrangements are unfeasible given the vast amount of permutations. Both the required network size and time to train the network would be unreasonably big. An alternative solution could be to train the model to recognize the components separately, i.e., the individual I/O modules. This knowledge can then be used to classify I/O compositions.

Another approach is to use unsupervised or self-taught learning as introduced by Raina et al. [17]. This would remove the need for manual classification of most of the training data, but would not stop the need for training on a large number of permutations, leading back to that the most feasible approach might be to make it able to detect and recognize the individual IO modules. Lampert et al. [18] describe a method of detecting unseen object classes by between-class attribute transfer. This approach would be interesting to assess with regards to the problem formulated in this paper. It would be quite easy to tag each label in our example with distinct attributes, e.g., number of configured modules, number of modules of a specific type.

A third approach is to not use ML at all but to use case-based reasoning [19] where one would formalize the description of an IO module and combinations of I/O modules.

V. Conclusions and Work-in-progress

In this paper, we present our work in progress towards a novel approach to classification of I/O configurations from recorded PROFINET I/O communication data utilizing neural networks. We show that the classification can be done with high accuracy. The work in this paper has as its primary purpose to demonstrate that it is possible to use ML techniques for this kind of tasks. However, the suggested method must be further developed towards being a method to reliably classify *any* configuration given the formulated problem. In our ongoing work we target analysis and assessment of the methods highlighted in the discussion, with the goal of being able to reliably perform classifications of data with a huge but well-described label-space.

References

[1] S. Zhao *et al.*, "Real-time network anomaly detection system using machine learning," in *DRCN*, 2015, pp. 267–270.

[2] N. Williams *et al.*, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 5–16, 2006.

[3] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[4] A. Kleinmann and A. Wool, "Automatic construction of statechart-based anomaly detection models for multi-threaded industrial control systems," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 4, pp. 55:1–55:21, 2017.

[5] K. Demertzis *et al.*, "A spiking one-class anomaly detection framework for cyber-security on industrial control systems," in *Engineering Applications of Neural Networks*, 2017, pp. 122–134.

[6] PI Organisation, "PROFINET," https://www.profibus.com, online; Accessed: 2019-03-19.

[7] ABB AB, "SelectIO," http://selectio.abb.com/, accessed: 2019-05-09.

[8] PI Organisation, "PROFIsafe," https://www.profibus.com/technology/profisafe/, online; Accessed: 2019-03-25.

[9] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

[10] K. Gurney, *An Introduction to Neural Networks*, 1997.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*, 1990, pp. 227–236.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[14] M. Jiang, Y. Liang, X. Feng, X. Fan, Z. Pei, Y. Xue, and R. Guan, "Text classification based on deep belief network and softmax regression," *Neural Computing and Applications*, vol. 29, no. 1, pp. 61–70, Jan 2018. [Online]. Available: https://doi.org/10.1007/s00521-016-2401-x

[15] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10).*, 2010.

[16] M. Du *et al.*, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017, pp. 1285–1298.

[17] R. Raina *et al.*, "Self-taught learning: Transfer learning from unlabeled data," in *ICML*, 2007, pp. 759–766.

[18] C. H. Lampert *et al.*, "Learning to detect unseen object classes by between-class attribute transfer," in *CVPR*, 2009.

[19] J. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, pp. 3–34, 03 1992.