Autoencoder Alignment Approach to Run-Time Interoperability for System of Systems Engineering

Jacob Nilsson, Jerker Delsing and Fredrik Sandin
EISLAB, Luleå University of Technology, 971 87 Luleå, Sweden
jacob.nilsson@ltu.se

Abstract—We formulate the challenging problem to establish information interoperability within a system of systems (SoS) as a machine-learning task, where autoencoder embeddings are aligned using message data and metadata to automate message translation. An SoS requires communication and collaboration between otherwise independently operating systems, which are subject to different standards, changing conditions, and hidden assumptions. Thus, interoperability approaches that are based on standardization and symbolic inference will have limited generalization and scalability in the SoS engineering domain. We present simulation experiments performed with message data generated using heating and ventilation system simulations. While the unsupervised learning approach proposed here remains unsolved in general, we obtained up to 75% translation accuracy with autoencoders aligned by back-translation after investigating seven different models with different training protocols and hyperparameters. For comparison, we obtain 100% translation accuracy on the same task with supervised learning, but the need for labeled training data makes that approach less interesting. We discuss possibilities to extend the proposed unsupervised learning approach to reach higher translation accuracy.

Index Terms—Adapter design pattern, Autoencoder alignment, Embeddings, M2M-Communication, System of systems

I. INTRODUCTION

The digitalization of society and production implies that different systems can be interconnected to improve flexibility and efficiency and to enable new services and products. In particular, this is the case in Industry 4.0 [1], [2] and the Internet of Things (IoT) [3], where different systems with sensors and actuators need to be interconnected. This requires purposeful communication among operational technology (OT) and information technology (IT) systems that otherwise operate independently and are subject to different standards, changing conditions, and hidden assumptions made by the engineers and users. The problem to enable such systems to interoperate and collaborate at runtime to meet higher-level goals through communication and emergent behavior is a core problem in System of Systems (SoS) engineering [4]–[8]. This calls for dynamic connectivity and interoperability among different autonomous systems, with interfaces and links forming and vanishing to enable the SoS to achieve and sustain its capabilities. Achieving this goal is a challenging problem, see [9]–[12] for references and further details.

The problem to automatically translate symbol and information representations between different domains of reference [13] is well studied in the field of natural language processing, where increasingly flexible and potent machine learning methods have set the standard for accurate translation over the last few years [14]. In the field of IoT and industrial automation, the work has focused on translation based on discrete symbolic metadata [15], [16], with relatively few [9], [10] recent investigations of subsymbolic data-driven methods, like the deep alignment of ontologies using auxiliary datasets [17].

It is common practice to engineer software adapters when interconnecting systems and components with different representational systems, to obtain a modular structure that makes testing and replacements of parts tractable. Searching for an efficient SoS engineering solution to the run-time (dynamic) interoperability problem, we investigate the feasibility of using autoencoders to generate adapters automatically in the form of encoder-decoder pairs operating on embeddings of the message information, like humans develop cognitive spaces [18] with associated perception and motor networks. Translation can then be enabled by aligning the embeddings of separate autoencoders through learning, a technique that has been used previously to align the embeddings of data belonging to different domains [19], and for style transfer of text [20], as well as for zero- and few-shot learning [21].

II. PROBLEM STATEMENT

The interoperability problem examined in this paper is defined in terms of cyber-physical systems (CPSs) [22], [23] that have some related degrees of freedom or measurable properties to be improved, and that send semantically related but syntactically different messages within each CPS, see [9] for further details. We investigate the feasibility of training autoencoders [24] to automatically generate translators for messages communicated between CPS taking part in a SoS.

An autoencoder is a neural network trained to reconstruct its input. Our model uses (a minimum of) two autoencoders, one for each of the message types that are being translated. In the basic form, an autoencoder generates an intermediary latent vector representation of the input, which typically has a lower dimension than the input while containing enough information to allow reconstruction of the input, see Figure 1a. By using the latent representations in this way they have a similar role as hand-made intermediary (“pivot”) representations, see for example [25].

In order to use the autoencoders for translation in this fashion, the latent representations produced by different encoders must be aligned, i.e. similar enough to ensure that...
Figure 1: Message autoencoding and translation with an aligned autoencoder. Each autoencoder features an encoder (Es), a decoder (Dx), and embeddings (ez) in a latent space. The CPSs co-exist in an environment, y = f(u). Aiming to improve some utility J(y) [9] an SoS is engineered by interconnecting the otherwise independently operating CPSs. (a) Independently trained autoencoders with essentially disjoint embeddings, which prevents correct message transcoding from CPS A to CPS B and vice versa. (b) Aligned autoencoders trained with additional constraints, like backtranslation loss and symbolic prior metainformation in the form of graphs G_A and G_B [9], which enables translation of messages via EA and DB (solid arrows), and via EB and DA (dotted arrows).

the different decoders produce messages containing the correct (purposeful) information. Here we use backtranslation loss to align the autoencoder latent representations. In backtransla-
tion, the model parameters are updated in two steps: First,
the messages of the two formats are autoencoded in a typical
manner. Second, the messages are transcoded into the other
format, transcoded back again, and the model parameters are
updated using the loss between the original messages and the
backtranslated messages, see the solid and dotted arrows in
Figure 1b. Backtranslation has been used in natural language
translation to generate translators [26].

Technically, the problem is how to train autoencoders so
that aligned latent representations needed to translate machine
messages are obtained, with messages that have varied struc-
ture and heterogeneous data types.

III. EXPERIMENTAL SETUP

We test the feasibility of learning aligned embeddings with
neural network autoencoders (AE) and backtranslation (BT)
with messages generated using a heating and ventilation sys-

Figure 2: (a) Room layout of simulation model. OO is outside, CC is the corridor, and the other rooms are offices. The layout has cyclic boundary conditions (dashed lines) (b) Simulated temperatures.

tem simulation, which enables us to generate parallel message
corpora for supervised and unsupervised learning experiments.

A. Simulation Model

We used a simulation that models the heat exchange be-
tween offices in a corridor with a varying outside temperature,
where each office has separate heating and cooling systems
with different target temperatures (setpoints). The temperatures
of each office, the corridor, and outside are monitored by two
temperature sensors, one for the heating system and one for
the ventilation system. The layout of the offices is illustrated
in Figure 2a, where the vertical dotted lines demonstrate
the cyclical boundary condition in which the eastern (NE
and SE) and western (NW and SW) offices are connected
wall to wall. Messages generated in the heating systems
form a set of comparable messages, which we reference with
A, and similarly, the messages generated in the ventilation
systems form a set of messages that we reference with B.

Heat exchange is implemented using the 1-dimensional heat
equation including the heat fluxes from neighboring rooms
and the outside, where the outside temperature is defined by
temperature data from the Swedish Meteorological Institute,
SMHI\(^1\). Figure 2b shows the simulated temperatures over
the course of the simulation.

The generated messages are encoded in JSON/SenML,
exemplified in Listing 1. Despite using the same encoding
and format, these messages represent the same information
differently. For A-type messages, the name "n" encodes
location and type of sensor, whereas the name "bn" in B-
type messages only encode type of sensor. Location is instead

\(^1\)www.smhi.se
Categorical fields
Continuous fields

The models use 1-hot encoding. Each categorical field is encoded using the longitude "Lon" and latitude "Lat" with a coordinate system specific to the simulation. The units "u" are different for both message types, and the values "v" have two modes: temperature in Kelvin (A-type) or degrees Celsius (B-type), and actuation in Watts (A-type) or percent of maximum power (B-type). 60000 randomly picked messages of each type were put in a training dataset, and another 20000 messages were put in validation and testing datasets, each containing 10000 messages.

To use these messages in a neural network, we transformed them from strings to vectors containing only the dynamic information. We separate the message fields into two kinds, categorical and continuous. The value fields are continuous, due to them representing a continuous variable, and the "t", "u", and "v" (A-type), "bn", and "bt" (B-type) are categorical fields because these fields can take one of a small number of discrete values. Each value field has a corresponding 1-hot vector representation, and the complete message vectors are concatenations of the 1-hot representations of the categorical values, and the value of the continuous fields. These vectors are illustrated in Figure 3.

B. Autoencoder with Backtranslation

The tested machine learning models are of two kinds, the first kind of model has one autoencoder (one encoder and one decoder) per message type, which we call a non-shared model. The second kind of model has encoders and decoders like the first, but those encoders and decoders share parameters, namely the layers closest to the latent representations. We call this a shared model. Sharing parameters like this has been shown to increase the performance of backtranslation strategies in natural language translation [27]. For reference, we also compare these unsupervised models to supervised models translating from format A to B. The supervised models also have the same encoder-decoder structure as the unsupervised models to make the comparison between the supervised and unsupervised models fair, and to test if translation is at all possible using this translation mechanism.

We vary the size of the encoders and decoders, between one layer of 10 parameters, or two layers of 10 and 9 parameters. The latent space has 8 dimensions across all tested models, which is more than sufficient to encode the messages considered here. The shared models always have three layers, which correspond in size to the innermost layers of the 2-layer model. During training, four outputs are produced: One autoencoded message $\hat{m}_{\text{auto}}$ and one backtranslated message $\hat{m}_{\text{back}}$ of type A and B respectively, see Figure 4.

C. Training Procedure

Model parameters are updated using three different strategies. The first is to update all parameters in both the autoencoding and backtranslation steps, which we call strategy 1. The second strategy is to update all parameters in the autoencoding step, and decoder parameters in the backtranslation step, which we call strategy 2. In strategy 3 the encoder parameters are updated in the backtranslation step, but the shared parameters are updated in both steps. Since strategy 3 uses the shared parameters, it is only tested on the shared model.

All unsupervised models use the same loss function

$$L = \frac{1}{2} \sum_{S=A,B} \frac{1}{N_{S_{\text{cat}}} + 1} \left( L^S_{\text{cat}} + L^S_{\text{con}} \right),$$

where $L^S_{\text{cat}}$ is the sum of the categorical cross-entropy loss for each categorical field in message type $S$ and $L^S_{\text{con}}$ is the mean-square loss for the continuous field message type $S$. Furthermore, the models are tested on the accuracy, i.e. the average accuracy for all categorical fields, and the mean square error of the continuous fields. All models use an Adam optimizer with Pytorch with a weight decay of 0.0001, with different learning rates between the supervised and unsupervised models. The supervised models use a flat learning rate of 0.01, and the unsupervised models used a cosine annealing with warm restarts schedule for the learning rate. Each epoch, the learning rate starts at 0.005 and drops to 0.0005 following a cosine curve. We used cosine annealing with warm restarts to minimize the risk of
TABLE I: Tested models and their results.

| Model | Kind      | Strategy | Size | Accuracy Max | Accuracy Mean | Accuracy Min | Error Max | Error Mean | Error Min |
|-------|-----------|----------|------|--------------|---------------|--------------|-----------|------------|-----------|
| 0     | non-shared| 2        | 1-layer | 0.70         | 0.44          | 0.57         | 4.0       |
| 1     | non-shared| 2        | 2-layer | 0.73         | 0.38          | 0.50         | 4.9       |
| 2     | non-shared| 1        | 1-layer | 0.66         | 0.39          | 0.48         | 6.7       |
| 3     | non-shared| 1        | 2-layer | 0.74         | 0.34          | 0.71         | 12.0      |
| 4     | shared    | 2        | 2-layer | 0.70         | 0.34          | 0.54         | 15.0      |
| 5     | shared    | 3        | 2-layer | 0.75         | 0.41          | 0.43         | 2.7       |
| 6     | shared    | 1        | 2-layer | 0.69         | 0.33          | 0.53         | 12.0      |
| 7     | supervised| –        | 1-layer | 1.0          | 1.0           | 0.16         | 0.17      |
| 8     | supervised| –        | 2-layer | 1.0          | 0.99          | 0.16         | 0.19      |

getting stuck in a local minimum, which often happened in the unsupervised case when using a flat learning rate.

In summary, we had 9 different models (see Table I) that were tested 30 times each with randomly initialized parameters to evaluate the statistics of their performance.

IV. RESULTS

Figure 5 shows boxplots of the highest categorical field translation accuracy (5a) and the lowest continuous field translation mean square error (5b) attained during validation for each of the 270 trained translators, organized by model. Table I also contains summary statistics of the models. It is clear that the unsupervised models performed much worse than the supervised models. Such a high performance of a supervised model is expected when trained on fairly simple data. The best performing model on average of the unsupervised models was the non-shared model using training strategy 2, but the widespread in both translation accuracy and translation MSE means we cannot definitely say that it was the best model. However, models trained with strategy 2 have a slightly higher categorical field translation accuracy and slightly lower continuous field translation error, perhaps because fewer parameters are being updated in each step in those models.

The large spread in best results is evidence of fragility in training, which is one of the main reasons we tested the shared models. However, it is evident that while the best results are promising in terms of translation accuracy (> 70%), the training procedure is still lacking in producing consistently good results for both tested values. This cannot be due to model capacity since the unsupervised model has the same shape as the unsupervised, the only influence must be the training procedure. Overall, we find the best results to be promising, but the training protocol needs to be made more robust for translation systems like these to ever be used in real-world scenarios.

V. DISCUSSION

While the average translation results are marginally better than pure chance, the best results demonstrate the feasibility of using unsupervised learning to train translators for message data. But the investigated training protocols are fragile and do not produce accurate translation results. In this use case, we can rule out model capacity as a source of error since the supervised model translates both categorical and continuous fields almost perfectly no matter the model size, therefore the issue should be related to the training protocol and not a priori the model itself. We suspected that one problem is that too
many parameters were updated at the same time, which is why training strategies 2 and 3 were considered. Figure 5 suggests that strategies 2 and 3 yields higher categorical accuracy and lower continuous MSE, which lends credibility to our suspicion that few parameters being updated lead to more robust training. To take this idea further, techniques from one-shot learning could be borrowed, like pre-training the autoencoder for one of the message types and then fine-tune it with the full dataset and backtranslation [27].

As explained in Section III, we hard-code the interface that transforms messages into vectors. This process is time-consuming, inflexible, and not suitable for dynamic SoS, due to the possible number of systems involved, and presents a problem if translators like these are to be used in production environments. One solution to this could be to use system metadata to inform the construction of the interface between the messages and encoders, e.g., the metadata could tell whether or not the fields are categorical or continuous. A more obvious solution would be to use a character-level recurrent neural network structure [28] to autoencode the messages, but this would not guarantee that the syntax of the reconstructed messages. Since JSON is defined by a context-free grammar, using a structure similar to the grammar-variational autoencoder [29] would be suitable for computer messages like these, since grammar-variational autoencoders guarantee that the syntax of the output is correct.

Still, guaranteeing the correct syntax of the message is not the same as guaranteeing that the data model will be translated correctly. Here, we can use the metadata instead to provide what fields are present in the message and use that information as input to the model. The metadata could for example be used as conditional input to a conditional generative model [30], [31] to increase the probability of correctly translating the data model. Alternatively, instead of including the metadata as an input to the model, the metadata could instead change the model structure. For example, we envision translators that are constructed by connecting various pre-trained modules, with the choice of which module and what part of a message it is fine-tuned on will depend on what metadata was used to pick that module. We conclude that some of the strategies proposed above are candidates to create a robust and flexible translation system to be used in future production SoS, and they are ideas that we want to delve deeper into in the future.

ACKNOWLEDGMENT

We would like to thank Dr. Sheraz Ahmed of the German Research Center for Artificial Intelligence, and Prof. Marcus Livicki of the Luleå University of Technology for helpful discussions that contributed to this work. This work is supported by the H2020 research and innovation programmes ECESEL Joint Undertaking and Vinnova, project no. 826452, and by The Kempe Foundations under contract SMK-1429. A preliminary version of this manuscript is included in the licentiate thesis by JN [32], however, the simulation experiments presented here are updated with further developed autoencoder architectures.
[23] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” Manufacturing Letters, vol. 3, pp. 18–23, 2015.

[24] I. J. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[25] H. Derhamy, J. Eliasson, and J. Delsing, “IoT interoperability—on-demand and low latency transparent multiprotocol translator,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1754–1763, 2017.

[26] S. Edunov, M. Ott, M. Auli, and D. Grangier, “Understanding back-translation at scale,” arXiv preprint arXiv:1808.09381, 2018.

[27] S. Benaim and L. Wolf, “One-shot unsupervised cross domain translation,” in Advances in Neural Information Processing Systems, 2018, pp. 2104–2114.

[28] J. Lee, K. Cho, and T. Hofmann, “Fully character-level neural machine translation without explicit segmentation,” Transactions of the Association for Computational Linguistics, vol. 5, pp. 365–378, 2017.

[29] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, “Grammar variational autoencoder,” in Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 1945–1954.

[30] K. Sohn, H. Lee, and X. Yan, “Learning structured output representation using deep conditional generative models,” in Advances in neural information processing systems, 2015, pp. 3483–3491.

[31] A. Mishra, S. Krishna Reddy, A. Mittal, and H. A. Murthy, “A generative model for zero-shot learning using conditional variational autoencoders,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2018, pp. 2188–2196.

[32] J. Nilsson, “System of systems interoperability machine learning model,” Ph.D. dissertation, Luleå University of Technology, 2019.