2021

Reliable and Interpretable Machine Learning for Modeling Physical and Cyber Systems

Daniel L. Marino Lizarazo
Virginia Commonwealth University

Reliable and Interpretable Machine Learning for Modeling Physical and Cyber Systems

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

by

DANIEL LEONARDO MARINO LIZARAZO

Automation Engineering, La Salle University, Colombia - 2015

Director: Milos Manic,
Professor, Department of Computer Science

Virginia Commonwealth University
Richmond, Virginia
November 2021

Acknowledgments

First, I would like to thank my advisor Prof. Milos Manic for his help, support, and guidance. I would like to thank Dr. Craig Rieger, Dr. David Shepherd, Dr. Preetam Ghosh, Dr. Ronald Boring, and Dr. Sergio Vazquez for serving in my dissertation committee and for their valuable feedback. Further, I would like to acknowledge the continuous support given to me by the Idaho National Laboratory (INL), specially for the data provided for the experimental section of chapter 4. I also want to acknowledge the support provided by the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation and workforce development.

I would also like to thank all the collaborators at other institutions and my colleagues for their help and support. I extend my gratitude to all the great colleagues of the Modern Heuristics Research Group, Chathuri, Morgan, Javi, Kasun, Billy, and Sandun for their support and contributions throughout my PhD program. Last but not least, I wish to thank my family and friends for their immense support during my academic journey. I dedicate my doctoral dissertation to my parents Antonio and Cristina, my brother Antonio, and my aunts Nona and Tere whose guidance, support, and values are the grounds that supported me through this journey.

Contents

List of Tables

List of Figures

Abstract

RELIABLE AND INTERPRETABLE MACHINE LEARNING FOR MODELING PHYSICAL AND CYBER SYSTEMS

By Daniel Leonardo Marino Lizarazo

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2021.

Director: Milos Manic,
Professor, Department of Computer Science

Over the past decade, Machine Learning (ML) research has predominantly focused on building extremely complex models in order to improve predictive performance. The idea was that performance can be improved by adding complexity to the models. This approach proved to be successful in creating models that can approximate highly complex relationships while taking advantage of large datasets. However, this approach led to extremely complex black-box models that lack reliability and are difficult to interpret. By lack of reliability, we specifically refer to the lack of consistent (unpredictable) behavior in situations outside the training data. Lack of interpretability refers to the lack of understanding of the inner workings of the learned models and the difficulty of extracting knowledge from trained models.

The objective of this dissertation is to improve the reliability and interpretability of ML models. In order to improve reliability, we focus on modeling uncertainty. We study the performance of several deterministic and stochastic Neural-Network models on modeling the dynamics of physical systems. We present an approach that uses Bayesian Neural Networks (BNNs) to model system dynamics under uncertainty. The BNN model is used to find optimal plan trajectories that take the system to a predefined state goal by estimating the state of the system several steps ahead. Modeling uncertainty using BNNs allowed us to improve the reliability of long-term estimations, increasing the performance of the planning task.

Interpretability is addressed by embedding domain-knowledge into data-driven models. We present an approach that embeds domain-knowledge extracted from physical laws into Variational Gaussian Processes. Domain-knowledge is embedded using a linear-prior that is derived from basic Newtonian Mechanics. We show that embedding domain-knowledge improves the predictive performance of the model. We also show that the approach provides an interpretable model without a degradation in accuracy.

In this dissertation, we center our discussion on modeling physical and cyber systems. Reliable and interpretable machine learning models are particularly sought in these applications, specially when they involve mission-critical and critical infrastructure operations. Reliable models in the presence of uncertainty are fundamental for the operation of physical and cyber systems. Modeling uncertainty provides a framework to inform users about possible incorrect predictions. In monitoring and diagnosis applications, interpretable models provide the means to extract useful information to operators. Finally, physical and cyber systems have rich structures with a large body of domain-knowledge that has yet to be explored in conjunction with machine learning systems. This dissertation shows how we can embed domain-knowledge from physical and cyber systems in order to improve interpretability. More specifically, we demonstrate how we can take advantage of physics domain-knowledge and the graph structure of cyber systems in order to design more structured and interpretable machine learning models.

The main contributions of this dissertation are: 1) we improve the reliability of Neural Network models in optimal planning tasks by modeling uncertainty; 2) we improve reliability and interpretability of data-driven system dynamics models by embedding domain-knowledge from physics; 3) we improve the interpretability of data-driven models of cyber systems by exploiting the graph structure of communication networks.

1. Introduction

Over the past decade, the field of Machine Learning (ML) has experienced a substantial growth powered, among other factors, by the availability of large datasets and the development of specialised hardware such as GPUs. The consolidation of the internet, including social networks and search engines, made huge quantities of data readily available for analysis and knowledge discovery. In order to deal with the increasing complexity and size of data, data scientists started to develop increasingly complex models. The idea was that increasing the complexity of models will automatically lead to improved performance. Suddenly, highly specialized hardware enabled data scientists and engineers to increase the complexity of ML models in order to handle the complexity found in large datasets. This led to an explosion in model complexity with unprecedented predictive performance. However, the increase in complexity also led to black-box, brittle models that lack trust. Perhaps the most prominent example of this trend has been the development of Deep Neural Networks [7].

In this dissertation, our objective is to improve reliability and interpretability. We approach reliability by modeling uncertainty. Uncertainty modeling is fundamental in order to improve reliability as it allows us to increase awareness of possible incorrect predictions. Uncertainty estimations help us to understand when and where we can (or can not) rely on the predictions made by data-driven models, warning users about regions where there is no enough information to make informed decisions. Uncertainty modeling also improves interpretability as it helps to answer questions like: why it failed? or is it going to fail?.

In this dissertation, we address interpretability by embedding domain-knowledge in machine learning. Domain-knowledge provides a way to improve interpretability and performance of machine learning models by taking advantage of available expert knowledge. Embedding domain-knowledge also serves as a tool to influence the inductive-bias of data-driven models, i.e. the assumptions made by the model in areas where data is not available.

1.1. Motivations

As ML technologies continue to permeate every industry and aspect of modern life, scientists, engineers, end-users, and governments are increasingly raising concerns about the reliability and interpretability of ML. As ML takes the lead in automated decision-making systems, it is imperative that we make it more transparent and reliable. This is fundamental in order to improve trust in these systems. Failure to meet these requirements may lead, in the best scenario, to a rejection of ML technologies in applications such as mission-critical systems and critical infrastructure [8]. In the worst case, it may lead to catastrophic consequences that can put in danger human lives [9], open opportunities for criminals to tamper systems [10], and create unfair and bias systems [8].

1.1.1. Motivation for Theoretical domain: trust, reliability, and interpretability

Trust

Recent years have seen rapid development in Machine Learning (ML). ML technologies percolate through every aspect of modern society, including healthcare, governments, industry, science, and education [11, 12, 13]. The way humans develop, deploy, and use these systems has changed through several decades, from handcrafted knowledge expert systems to systems with excellent capability to perceive, learn, and make decisions [14, 15, 16]. However, despite the remarkable advancements, there is a

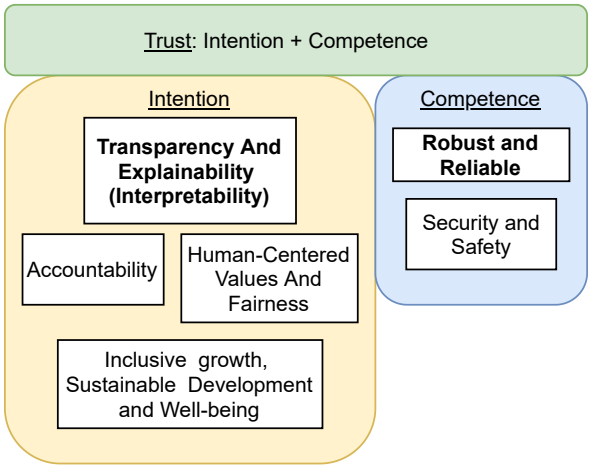


Figure 1.1.: Mapping of Intention and Competence for the Trustworthy AI Principles proposed by the OECD, G20, the White House, and Deloitte [2, 3, 4]

reluctance from some sectors to adopt ML because of concerns that are ultimately related to a lack of trust in these systems.

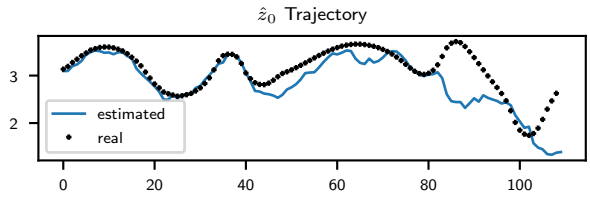
Although there is no agreement on the exact definition of *trust*, in general, trust can be conceptualized as predictable behavior, even in the presence of uncertainty [17]. This means a human trusts an agent when the behavior of the agent meets some expectations, even when dealing with unpredictable circumstances.

Trustworthy Artificial Intelligence (AI) research area focuses on improving human *trust* in AI/ML systems. Trustworthy AI has been defined as a combination of diverse research areas, which includes fairness, robustness, explainability, accountability, verifiability, transparency, and sustainability of AI systems [18, 2, 19, 3, 20]. The goal of Trustworthy AI research is to: 1) identify factors which harm the human trust of AI systems, and 2) introduce methods to improve human trust in AI systems [21]. Trustworthy AI has emerged in recent years as a core objective of governments, companies, and researchers [18]. As shown in Fig. 1.1, Trustworthy AI is an umbrella framework that incorporates many disciplines and subjects. In this dissertation, we specifically address reliability and interpretability.

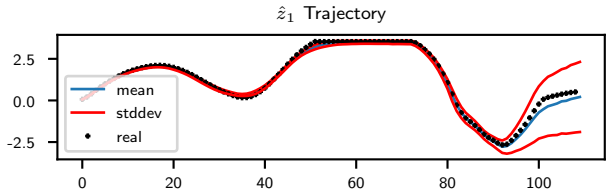
Trust can be conceptualized as the combination of two main components: *Intention* and *Competence* [22]. Both Intentions and Competence are fundamental in order to assess the predictability of the agent under uncertain circumstances. Describing trustworthiness as the assessment of Intention and Competence allows us to understand the key discussion points about Trustworthy AI. Fig. 1.1 shows the core research areas for Trustworthy AI defined by the OECD [18, 3, 2, 19]. The figure shows how we can group these areas as promoting either intention or competence.

Concerns regarding *Competence* are primarily linked to reliability, robustness, security, and safety issues of ML systems. As an example, using Deep Learning systems in critical applications has been challenging due to the low reliability and robustness of the networks [10]. Deep Neural Networks (DNNs) can be fooled by malicious adversaries using adversarial samples, where slight modifications in the input of ML algorithms can cause incorrect classifications. This behavior has also been evidenced in physical world scenarios under normal operation regions [23], becoming one of the major risks for applying DNNs in safety-critical environments [24].

Concerns regarding *Intention* can be linked to a lack of *Transparency and Explainability* of state-of-the-art ML technologies. When ML systems become excellent in their performance, their complexity also increases, making them harder to interpret. A clear example of this trend can be seen on the black-box perception of technologies such as Deep Neural Networks [25, 26]. Users hesitate to trust these ML systems because they can not understand the decision-making process [27]. The area of Explainable



(a) Unreliable: the estimated trajectory deviates from the real trajectory from t=45



(b) Reliable: the estimated trajectory stays close to the real trajectory, plus the uncertainty estimation warns of possible incorrect outcomes.

Figure 1.2.: Unreliable vs Reliable model: estimation of a system state trajectory over time.

AI [28] was introduced to open the black-box of Neural Networks and understand the decision making process of ML systems. Explanations methods include but not limited to global model explanation [26], local interpretations [25], and interactive visualization. However, effectiveness of explainability methods mainly depend on the context of the problem. Specifically, for high-risk areas like medical diagnostic, transportation, and robot teleportation, there is a reluctance to remove humans entirely from the loop [29].

In the following sections we expand on the reliability and interpretability motivations of the dissertation. We present the definitions for interpretability and reliability relevant to this dissertation.

Reliability

There is certain ambiguity when using the concepts of robustness and reliability in regards of Trustworthy AI. In order to prevent ambiguity, we use the definition of correctness, robustness, and reliability provided by [30, 31]:

- ***Correctness****: The degree to which a system is free from faults in its specification, design, and implementation. [30]*
- ***Robustness****: The degree to which a system continues to function in the presence of invalid inputs or stressful environmental conditions. [30]*
- ***Reliability****: The ability of a system to perform its required functions under stated conditions whenever required—having a long mean time between failures [30]. Reliability is a statistical approximation to correctness, in the sense that 100% reliability is indistinguishable from correctness [31].*

Traditional ANNs are often evaluated on the accuracy of its predictions by splitting the data in training and testing datasets. This approach provides a basic way of measuring the reliability of ANNs by evaluating the accuracy and failure rates of the model. This approach also provides a way to

guard against overfitting and improve generalization by keeping training and testing datasets separate. However, the success of this approach is heavily influenced by the way training and testing datasets are sampled. This approach is successful if the training and testing dataset contains data from all regions in the sample space. This is an assumption that is not trivial to guarantee in practice. In many applications, it is common to have unexplored regions due to sampling bias, system constraints, or simply because it is too expensive to collect data. Because training and testing datasets are usually collected using the same methodology, unexplored regions in training data are also present in testing data, meaning that testing datasets are not a guarantee for reliability. Black-box, deterministic models such as ANNs do not define any consistent behavior in areas where training/testing data is not available. This means that the outputs of the model in unexplored regions are not accounted for, posing a serious reliability issue as there is no way to evaluate the correctness in these regions.

Unexplored regions are one of the many sources of uncertainty that machine learning models will encounter within normal operating conditions. By not accounting for these uncertainties, traditional deterministic ANNs are prone to failures as there is no way to inform users of possible invalid outputs. This is exemplified in Fig. 1.2. Fig. 1.2a shows an example of an unreliable machine-learning model. In this example, the model is used to provide an estimation of a system state trajectory. Process noise and approximation errors cause long-term estimations to deviate from the real behavior of the system. This deviation of the estimations poses a serious reliability problem, as the model cannot be relied to provide long-term estimations.

In order to improve reliability, we focus on modeling uncertainty using data-driven stochastic models. Traditional ANNs lack consistent behavior in areas where training/testing data is not available. ANNs are powerful models for interpolation, while performance in extrapolation is often left to chance. Modeling uncertainty allows us to specify a desired consistent behavior when in the presence of uncertainty. Fig. 1.2b shows an example of a reliable model. This model provides an estimation of the uncertainty, providing a reliable long-term estimation of the state trajectory.

In this dissertation, we use the reliability term in the broad sense, based on the definition and discussion provided above. The analysis that we present is not related to reliability engineering. Furthermore, we consider failures only pertaining to *reliability* and not *robustness*. Robustness evaluates the functionality of the system when in the presence of invalid inputs or stress conditions. The failures considered in this dissertation can be attributed to uncertainties that are present during the normal operation of the ML model.

Interpretability and Expert-Bias

Interpretability in machine learning focuses on improving models in two main fronts[32]:

- **Transparency**: refers to the ability of the user to understand the inner workings of the model.
- **Post-hoc explanations**: refer to the ability to extract information from the learned model.

The need for interpretability can be attributed to the imperfect nature of Machine Learning models. Interpretability is specially sought when the output of a model does not conform to the expectations of the user. ML models are at their core an *approximation* of real-world processes. This means that estimation errors are part of the normal operation of these systems. Understanding the inner workings of a model can help to identify and correct failures. Depending on the application, interpretability may be required in order to assess the legality or establish accountability for a ML-based decision-making process [2, 32]. Interpretability is also a common requirement on diagnosis systems, where identifying a fault or an anomaly is as important as to identify the origin and cause. Interpretable models offer an option to get more useful information from the model.

Interpretability is also a tool that can be used to manage and assess bias. Bias has gained attention in recent years, usually with a negative connotation as it often leads to unfair systems. For example, sample

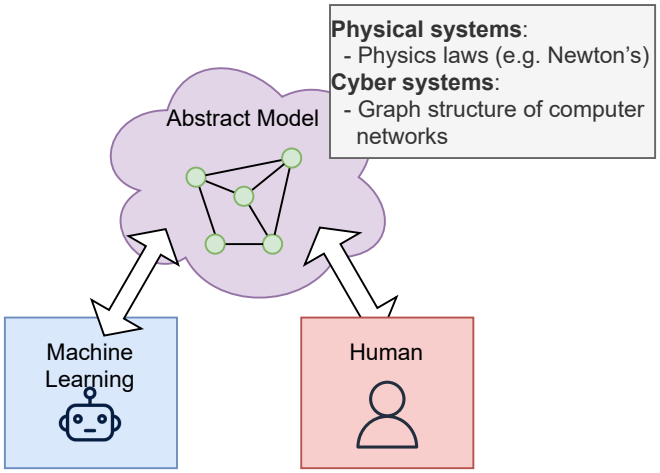


Figure 1.3.: Abstract model as a shared representation to bridge the gap between ML and Human understanding of a problem.

bias can significantly skew correlations, leading to incorrect assumptions in sample groups with poor representation. On the other hand, bias is also a fundamental mechanism used by Machine Learning (ML) models to perform inference in previously unseen situations. *Inductive bias* is a property of every ML model that refers to the assumptions made by the model when making predictions on novel data. Inductive bias is the core mechanics that enables ML models to generalize to previously unseen data. In this sense, bias is fundamental to define the behavior of the ML model when extrapolating from previously seen scenarios. As such, bias is not inherently bad, but it needs to be properly controlled and managed. The most dangerous bias in AI is typically the one that is unaccounted for, i.e., the one we don't know about. Black-box models accentuate problems with unwanted bias as they provide very limited options to understand the how and why of the decision making process [8]. On the other hand, interpretable models have as objective to improve the understanding of cause and effect.

Inductive biases can help to introduce assumptions about the real process being modeled or the space of valid solutions for the problem at hand. From a Bayesian probability perspective, inductive bias is expressed in the form of prior probability distributions [33]. These distributions describe the assumptions made by the model before taking in consideration the available data. A clear understanding of the inductive bias and prior distributions of an ML model is essential in order to understand their decision-making process.

In this dissertation, we approach interpretability by incorporating domain-knowledge from physical and cyber systems into machine learning models. Engineers have spent centuries constructing abstract models of physical and more recently cyber systems, accumulating a large volume of domain-knowledge that has jet to be fully explored for the development of ML models. These abstractions have been built in order for humans to understand the world and are the basis of all engineering disciplines. By incorporating these abstract models as part of the inductive bias of ML models, we can create a bridge between the human understanding of a system and the assumptions that the ML model uses for decision making (Figure 1.3). The abstract model serves as a proxy for the user to interact with the ML model, providing common ground with the objective of strengthening the understanding and trust on the ML model. This approach allows us to improve the inductive bias of the ML model by taking advantage of the inherent structure of the target problem. The presented approaches aim to improve the transparency of the models by creating more interpretable and structured models [34].

Figure 1.4.: Foundations of SciML, US DoE Priority Research Directions [5].

1.1.2. Motivation for Application domain: modeling physical and cyber systems

Applying Machine Learning to model physical and cyber systems has gained attention in recent years. Several US DOE initiatives have been funded over the past two decades to use Machine Learning to enhance modeling and simulation [5]. Intelligent automation and decision support have also been an area of interest. ML models of physical systems provide a tool that can be used for forecasting [35], control [36], decision support [37], and monitoring [38]. ML models are also increasingly used for modeling cyber systems. Some ML applications include anomaly detection for monitoring computer networks [38] and optimizing network traffic [5]. Nowadays, mission-critical infrastructure is composed of a tight combination of cyber and physical components that integrate computational resources, communication, control, and physical processes into a single system [39, 40].

Trustworthy ML models are specially sought in applications for physical and cyber systems, specially when it involves mission-critical systems and critical infrastructure. The increase of complexity of ML models and their black-box perception has lead to a change in objectives in ML research. Instead of focusing on predictive performance, recent research efforts are specially focused on developing more transparent and reliable models. This trend can be evidenced by the creation of several programs to tackle transparency and reliability sponsored by several US federal agencies, including the US department of energy, department of defense, the National Science Foundation, among others [34, 41, 3]. These programs have been included in several national strategic plans such as the Darpa Explainable AI Program [34], The National Artificial Intelligence Research And Development Strategic Plan [41], and the American Artificial Intelligence Initiative [3].

Fig. 1.4 shows the priority research directions for Scientific Machine Learning, laid down by the US DOE [5]. The first area entails leveraging domain knowledge which includes the use of physical principles and physics-informed priors. Domain-knowledge also entails exploiting the inherent structures of each specific problem. Interpretability and reliability are shown as a central part of the other two foundation areas. Uncertainty quantification is included in the interpretable foundational area as it provides tools to understand failures of ML models. Probabilistic modeling in ML is included as part of the robust and reliable foundational area as it provides a framework to deal with uncertainties. In this dissertation, we use Machine Learning algorithms to model physical and cyber systems. We follow the research priority directions of SciML closely by modeling uncertainty, incorporating domain knowledge from physics, and taking advantage of the structures in cyber systems.

1.2. Goals and Contributions

Objective: Improve the reliability and interpretability of machine learning for modeling physical and cyber systems.

1. **Goal**: To improve the reliability of Neural Networks in optimal planning tasks by modeling uncertainty.
   Contribution 1: A methodology for modeling and planning under uncertainty using Bayesian Neural Networks of system dynamics
   - Improving the reliability of long-term modeling and planning tasks using Bayesian Neural Networks.
   - Two approaches for learning system dynamics: using single-step predictions and using multi-step predictions.
   - An analysis of two approaches to model uncertainty: homoscedastic and heteroscedastic modeling.
2. **Goal**: To improve interpretability of machine learning models of physical systems by embedding domain-knowledge.
   Contribution 2: The Explicit Variational Gaussian Process model, a stochastic data-driven model enhanced with domain-knowledge from physics.
   - Improving interpretability and reliability while using less amount of data when compared with fully data-driven approaches.
   - Improving accuracy using simple physics-based priors, demonstrating that accuracy improves even when using heavily simplified physics as domain-knowledge.
   - Scalability to large datasets using the sparse variational framework.
   - Improving interpretability using the posterior distribution of trainable parameters, i.e. the value of the model parameters after training.
3. **Goal**: To improve the interpretability of machine learning models of cyber systems by exploiting the graph structure of computer networks
   Contribution 3: The Network Transformer, a self-supervised Neural Network model that improves interpretability of cyber anomaly detection models by leveraging graph-based ML representations.
   - Interpretable model using graph-based ML representations that incorporate the structure of the computer network.
   - Self-supervised data-driven discovery of representative communication patterns for anomaly detection
   - Scalability of the presented approach to large datasets

1.3. Organization of the Dissertation

The rest of the dissertation document is organized as follows:

- Chapter 2: presents goal 1, contribution 1, a methodology to improve the reliability of long-term planning using Neural Network models of system dynamics. The chapter presents a methodology for modeling and planning under uncertainty using Bayesian Neural Networks (BNNs).

- Chapter 3: presents goal 2, contribution 2, the Explicit Variational Gaussian Process model, a stochastic data-driven model enhanced with domain-knowledge from physics. The approach allows to take advantage of domain-knowledge to improve interpretability and reliability while using less amount of data when compared with black-box approaches.
- Chapter 4: presents goal 3, contribution 3, the Network Transformer, an interpretable model for anomaly detection in cyber systems. This model introduces a self-supervised way to profile the normal communication patterns and identify anomalous behavior using a hierarchical set of features. The approach incorporates the graph structure of the communication network in order to perform anomaly detection. The model allows to identify anomalies and identify the devices and connections affected by the anomalies.
- Chapter 5: concludes the dissertation and presents a discussion of current trends and future opportunities.

2. Improving reliability of Neural Networks in optimal planning by modeling uncertainty

This chapter is in support of Contribution 1: *A methodology for modeling and planning under uncertainty using Bayesian Neural Networks of system dynamics*

- *Improving the reliability of long-term modeling and planning tasks using Bayesian Neural Networks.*
- *Two approaches for learning system dynamics: using single-step predictions and using multi-step predictions.*
- *An analysis of two approaches to model uncertainty: homoscedastic and heteroscedastic modeling.*

Paper:

-

Papers that preceded this work:

- D. Marino, K. Amarasinghe, M. Anderson, N. Yancey, Q. Nguyen, K. Kenney, M. Manic, "Data-driven decision support for reliable biomass feedstock preprocessing" , IEEE Resilience Week (RWS) 2017, Wilmington, DE, USA, Sep, 18-22, 2017
- D. Marino, K. Amarasinghe, M. Manic, "Building Energy Load Forecasting using Deep Neural Networks," in Press. 42nd Annual Conference of the IEEE Industrial Electronics Society, IEEE IECON 2016, Florence, Italy, Oct. 24-27, 2016

Artificial Neural Networks (ANNs) have been frequently used in industrial applications to model complex systems. However, using traditional ANNs for long-term planning tasks remains a challenge as they lack the capability to model uncertainty. Process noise and approximation errors cause ANN long-term estimations to deviate from the real behavior of the system. This deviation of the estimations pose a serious reliability problem, as long-term estimations cannot be trusted to find optimal control actions using traditional ANN models. Unlike traditional ANNs, stochastic models provide a natural way to model uncertainty, providing estimations over a range of several possible outcomes. This chapter introduces a stochastic modeling and planning approach using Deep Bayesian Neural Networks (DBNNs). We use DBNNs to learn a stochastic model of the system dynamics. Planning is addressed as an open-loop trajectory optimization problem. We present two approaches for learning the dynamics: using single-step predictions and using multi-step predictions. The advantages of the presented methodology are: 1) accurate long-term estimations of the system state-trajectory probability distribution without the need for expert knowledge of the dynamics; 2) improved generalization and faster convergence rates in the trajectory optimization task when using multi-step predictions to train the model; 3) viable for real-world applications since all expensive optimizations are executed offline while using a reasonable

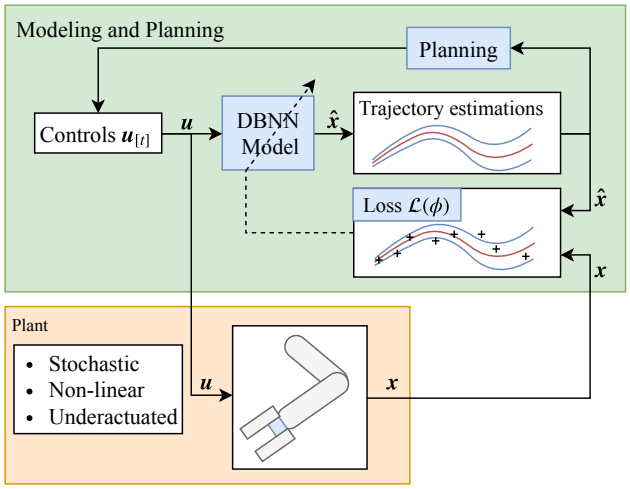


Figure 2.1.: Overview: Modeling and planning using DBNNs.

number of data-samples. Testing is performed using underactuated benchmark problems: the Cartpole and Acrobot. The presented methodology successfully learned the swing-up maneuver using a relatively small number of iterations, with less than 125 sampled trajectories, and without any expert knowledge of the dynamics [42].

2.1. Introduction

Control of underactuated non-linear systems is often broken down into three tasks [43] [44]: modeling system dynamics, open-loop trajectory planning, and trajectory stabilization. Historically, modeling has been performed using first principles. However, this approach is labor intensive and requires highly skilled professionals. A promising alternative is to use data-driven models to learn the system dynamics. Fully data-driven models provide a flexible approach that does not require the derivation of complex mathematical models from first principles, reducing the need for expert knowledge [45].

Deterministic neural-networks have been extensively used in the control of industrial systems, some applications include: controlling electric drives [46] [47], robot manipulators [48] and HVAC systems [49]. Despite the universal approximation properties of ANNs, standard deterministic ANNs lack the capacity of modeling uncertainty. As a consequence, deterministic ANNs are sensitive to model-mismatch problems where long-term estimations deviate from the behavior of the real system [45] [50]. Because of the inherent stochastic nature of real systems and the approximation errors introduced by data-driven modeling, we cannot expect an algorithm to provide long-term predictions with 100% accuracy. A more realistic approach is to design a model that provides accurate long-term predictions of the probability distribution of the system state trajectory. The estimations should take into account the uncertainties of both the plant and the learned model.

Stochastic data-driven models provide a natural way to model uncertainty, which can be used to mitigate model-mismatch problems [45]. These models allow us to make informed decisions using the predictions of the model while being cautious about the uncertainty of such predictions [37].

In this chapter, we address the first two stages of controlling underactuated systems: modeling system dynamics and open-loop trajectory planning. Our methodology uses a Deep Bayesian Neural Network (DBNN) to learn a stochastic model of the system dynamics. The learned model is used to estimate the probability distribution of long-term system state trajectories. Planning is addressed as an open-loop trajectory optimization problem, where long-term trajectory estimations are used to find optimal

control inputs that accomplish a given task. Fig. 2.1 shows an overview of the methodology.

In this chapter, we mainly focus on uncertainties derived from the approximation nature of the learned model. Uncertainties in this chapter arise from: 1) unexplored areas where data is not available; 2) approximation errors of the DBNN model; 3) time discretization; 4) uncertainty from numerical computations in the simulation.

The rest of the chapter is organized as follows: Section 2.2 provides a background of DBNNs and its training using variational inference. Section 2.3 describes the trajectory optimization task and how the system dynamics are learned using single-step and multi-step predictions. Section 2.4 presents the experimental evaluation. Section 2.5 describes the related work on the area of data-driven modeling and control. Section 2.6 concludes the chapter.

2.2. Stochastic modeling using Bayesian neural-networks

This section presents a brief overview of training DBNNs using variational inference [51] [52]. Given a dataset $\mathcal{D} = \left\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) | \boldsymbol{y}^{(i)} \in \boldsymbol{Y}, \boldsymbol{x}^{(i)} \in \boldsymbol{X}\right\}_{i=1}^{|\mathcal{D}|}$ of input/output pairs of samples $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, we would like to learn a probability distribution $p(\boldsymbol{y}|\boldsymbol{x}, w)$, parameterized by w, that would allow us to make predictions over previously unseen test points $\boldsymbol{y}^*, \boldsymbol{x}^*$.

In the Bayesian setting, the probability distribution for a test point $\boldsymbol{x}^*$ can be expressed as follows:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{Y}, \boldsymbol{X}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, w) p(w|\boldsymbol{Y}, \boldsymbol{X}) dw \tag{2.1}$$

where $p(w|\boldsymbol{Y}, \boldsymbol{X})$ is the probability of w given the training dataset $\boldsymbol{Y}, \boldsymbol{X}$. In other words, $p(w|\boldsymbol{Y}, \boldsymbol{X})$ quantifies how well a given value of w represents the data sampled from the real system. Using Bayes theorem, this distribution is defined as follows:

$$p(w|\boldsymbol{Y}, \boldsymbol{X}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X}, w) p(w)}{p(\boldsymbol{Y}|\boldsymbol{X})}$$

where $p(w)$ is the prior probability for the parameters of the model. In practice, the prior is used to prevent over-fitting.

As $p(w|\boldsymbol{Y}, \boldsymbol{X})$ is typically intractable, an approximation $q_\phi(w)$ is used in practice [52]. The Kullback-Leibler (KL) divergence is used to measure the difference between these distributions. The objective then becomes to find the parameters ϕ^* of the distribution $q_\phi(w)$ that minimize the difference between the distributions $p(w|\boldsymbol{Y}, \boldsymbol{X})$ and $q_\phi(w)$:

$$\phi^* = \arg\min_\phi KL\left(q_\phi(w) \| p(w|X, Y)\right) \tag{2.2}$$

By solving the minimization problem in Eq. (2.2), we find the distribution $q_{\phi^*}(w)$ that best approximates $p(w|\boldsymbol{Y}, \boldsymbol{X})$. From the definition of the KL divergence, we can obtain a more convenient way to express Eq. (2.2):

$$\begin{aligned} KL\left(q_\phi(w)|p(w|\boldsymbol{X}, \boldsymbol{Y})\right) &= \int_w q_\phi(w) \ln\left(\frac{q_\phi(w)}{p(w|\boldsymbol{X}, \boldsymbol{Y})}\right) dw \\ &= \underbrace{\left[\mathop{\mathbb{E}}_{w\sim q_\phi(w)}\left[-\ln\left(p(\boldsymbol{Y}|\boldsymbol{X}, w)\right)\right] + KL\left(q_\phi(w)\|p(w)\right)\right]}_{\text{negative log evidence lower bound } \mathcal{L}(\phi)} + \ln\left(p(\boldsymbol{Y}|\boldsymbol{X})\right) \end{aligned}$$

Given that $P(\boldsymbol{Y}|\boldsymbol{X})$ is constant (does not depend on ϕ), minimizing the negative *log evidence lower bound* (ELBO) $\mathcal{L}(\phi)$ is equivalent to minimizing the KL divergence in Eq. (2.2) [51][52].

Assuming our dataset $\mathcal{D}$ is composed of $|\mathcal{D}|$ number of independent and identically distributed samples the loss w.r.t. the variational parameters ϕ can be expressed as follows:

$$\mathcal{L}(\phi) = \sum_{i=1}^{|\mathcal{D}|} \mathop{\mathbb{E}}_{q_\phi(w)} \left[-\ln \left(p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)}, w) \right) \right] + KL\left(q_\phi(w) \| p(w)\right) \tag{2.3}$$

In sections 2.3.1 and 2.3.2, $\mathcal{L}(\phi)$ will be used as the loss for learning systems dynamics, quantifying how well the model approximates the data sampled from the real system. The loss $\mathcal{L}(\phi)$ is composed of two main terms: $\mathbb{E}_{w \sim q_\phi(w)} \left[\ln \left(p(\boldsymbol{Y} | \boldsymbol{X}, w) \right) \right]$ measures how well the data-driven model fits the data; $KL\left(q_\phi(w) \| p(w)\right)$ usually works as a regularizer, and often takes the form of an l_2 regularization of the neural-network weights (See Appendix B.1).

We can optimize Eq. (2.3) using a Monte-Carlo approach to approximate the expectation. Furthermore, back-propagation can be used to calculate the gradients w.r.t. ϕ if $q_\phi(w)$ is represented using the re-parameterization trick [53]. Dropout is an example of a simple way to re-parameterize a variational distribution $q_\phi(w)$ that takes the form of a Mixture of Gaussians[52].

A common distribution adopted for $p(\boldsymbol{y}|\boldsymbol{x}, w)$ is a multivariate Gaussian distribution with diagonal precision $\hat{\boldsymbol{\tau}}$:

$$p(\boldsymbol{y}|\boldsymbol{x}, w) = \mathcal{N}\left(\boldsymbol{y} \mid \hat{\boldsymbol{\mu}}(\boldsymbol{x}, w_\mu),\ \mathrm{diag}\left(\hat{\boldsymbol{\tau}}(\boldsymbol{x}, w_\tau)\right)^{-1} \right) \tag{2.4}$$

where $\hat{\boldsymbol{\mu}}(\boldsymbol{x}, w_\mu), \hat{\boldsymbol{\tau}}(\boldsymbol{x}, w_\tau)$ are functions parameterized by $w = w_\mu \cup w_\tau$. $\hat{\boldsymbol{\mu}}$ approximates the mean of the Gaussian distribution, while $\hat{\boldsymbol{\tau}}$ approximates the reciprocal of the variance.

Using Eq. (2.4) as our model, the loss in Eq. (2.3) can be estimated as follows:

$$\mathcal{L}(\phi) \approx \frac{1}{2\,|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \mathcal{L}_R\left(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)} \right) + \frac{1}{|\mathcal{D}|} KL\left(q_\phi(w) \| p(w)\right) \tag{2.5}$$

where

$$\begin{aligned} \mathcal{L}_R\left(\boldsymbol{y}, \boldsymbol{x}\right) &= -\log(\hat{\boldsymbol{\tau}}_{\boldsymbol{x}})^T \mathbf{1} + \left\| \sqrt{\hat{\boldsymbol{\tau}}_{\boldsymbol{x}}} \odot \left(\boldsymbol{y} - \hat{\boldsymbol{\mu}}_{\boldsymbol{x}} \right) \right\|^2 \\ \hat{\boldsymbol{\mu}}_{\boldsymbol{x}} &= \hat{\boldsymbol{\mu}}(\boldsymbol{x}, w_\mu); \quad w \sim q_\phi(w) \\ \hat{\boldsymbol{\tau}}_{\boldsymbol{x}} &= \hat{\boldsymbol{\tau}}(\boldsymbol{x}, w_\tau) \end{aligned}$$

We use $\odot$ to denote the element-wise (Hadamard) product. $\mathbf{1}$ represents a vector filled with ones. Note that Eq. (2.5) is a Monte-Carlo approximation of the loss, using parameters w sampled from the variational distribution $w \sim q_\phi(w)$.

Having obtained the parameters ϕ^* that minimize Eq. (2.5), the distribution $q_{\phi^*}(w)$ can be used to estimate Eq. (2.1) (the test prediction) using:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{Y}) \approx \mathop{\mathbb{E}}_{w \sim q_{\phi^*}(w)} \left[p(\boldsymbol{y}^*|\boldsymbol{x}^*, w) \right]$$

Depending on how $\hat{\boldsymbol{\tau}}$ is defined, we can use Eq. (2.4) to model homoscedastic and heteroscedastic distributions. Homoscedastic models assume the variance is constant, while heteroscedastic models assume the variance changes depending on the inputs $\boldsymbol{x}$. Fig. 2.2 shows an example of an heteroscedastic distributions modeled using DBNNs. This chapter makes emphasis on heteroscedastic models, as they are more general, challenging and commonly found in industrial systems [54].

In this chapter, $\hat{\mu}(\boldsymbol{x}, w_\mu)$ is modeled using a multi-layer perceptron with dropout. For homoscedastic models, the precision is defined as a constant $\hat{\tau}(\boldsymbol{x}, w_\tau) = \boldsymbol{\tau}_w^2$, which is jointly optimized when minimizing Eq. (2.5). For heteroscedastic models, $\hat{\tau}(\boldsymbol{x}, w_\mu)$ is a multi-layer perceptron without dropout. Appendix B.1 describes in detail the variational distributions and priors used to model the parameters.

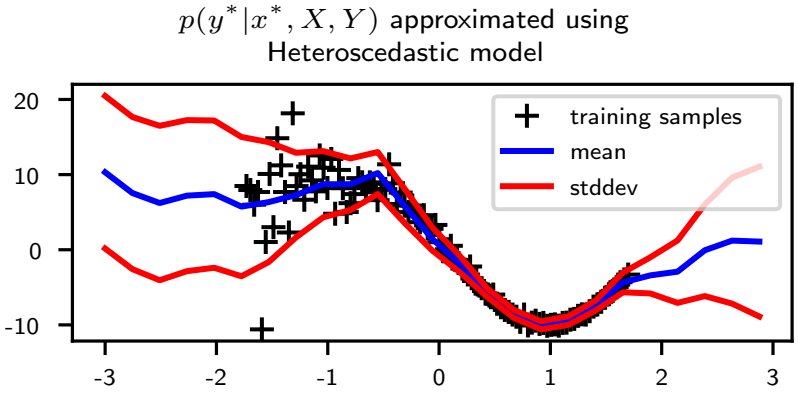


Figure 2.2.: Heteroscedastic distribution learned using a Bayesian neural-network. The estimated standard deviation shows how the uncertainty grows as we move away from the training dataset

2.3. Modeling and planning using DBNNs

This section describes the presented approach for modeling and planning under uncertainty using Deep Bayesian Neural-Networks (DBNNs). Specifically, we present how DBNNs are trained to approximate non-linear dynamic systems using single-step and multiple-step predictions. Planning is formulated as a trajectory optimization problem, where controls are applied in open-loop.

In this work, we consider fully observable dynamic systems. We assume a stochastic dynamic system model that takes the following form:

$$\boldsymbol{z}_{[t+1]} = \boldsymbol{z}_{[t]} + \Delta_{[t]} \tag{2.6}$$

$$\Delta_{[t]} \sim \mathcal{N}\left(\boldsymbol{\mu}_{[t]}, \operatorname{diag}\left(\boldsymbol{\tau}_{[t]}\right)^{-1}\right) \tag{2.7}$$

$$\boldsymbol{\mu}_{[t]} = \boldsymbol{\mu}\left(\boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]}, w_{\mu}\right)$$

$$\boldsymbol{\tau}_{[t]} = \boldsymbol{\tau}\left(\boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]}, w_{\tau}\right)$$

where $\boldsymbol{z}_{[t]}$ is the state of the system at time step t, and $\Delta_{[t]}$ is the increment when the control signal $\boldsymbol{u}_{[t]}$ is applied on the system. The symbol $\oplus$ represents the concatenation operator. $\Delta_{[t]}$ is modeled using the DBNN described in Eq. (2.4), with inputs/outputs defined as $\boldsymbol{x} = \boldsymbol{z} \oplus \boldsymbol{u}$ and $\boldsymbol{y} = \Delta$. Fig. 2.3a shows the architecture of the DBNN.

Modeling $\Delta_{[t]}$ using a Gaussian distribution serves the purpose of explicitly including a mechanism to handle unmodeled disturbances (process noise). The variational posterior placed on the parameters serves the purpose of dealing with the uncertainties from the learned model, increasing uncertainty in areas where data is not available. In this work, we assume zero measurement noise. Note that in Eq. (2.6) the noise derived from $\Delta_{[t]}$ is propagated over time. This formulation ensures that uncertainty increases over time.

The trajectory optimization problem in this chapter is defined as a finite-horizon open-loop control problem with deterministic control inputs $\boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}$. Finite-horizon problems look for a set of optimal inputs $\boldsymbol{u}_{[t]}$ that minimize a control cost that depends on the state of the system for a finite number of time steps. The control cost quantifies how effective the inputs are in driving the system towards completing a particular task. In this work, we only consider open-loop control, which means that the control signal $\boldsymbol{u}_{[t]}$ does not depend on the current state $\boldsymbol{z}_{[t]}$. In contrast, closed-loop approaches look for control policies that depend on the current state of the system (i.e. $\boldsymbol{u}_{[t]} = g(\boldsymbol{z}_{[t]})$).

Concretely, the objective of the trajectory optimization problem is to find the control inputs $\boldsymbol{u}_{[t]}$ that minimize the control cost $\mathcal{L}_c$:

$$\min_{\{\boldsymbol{u}_{[1]},...,\boldsymbol{u}_{[T_c]}\}} \mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right)$$

where $\boldsymbol{z}_{[t]}$ is the state of the system at time step t, $\boldsymbol{z}_{[1]}$ is the initial state of the system, and the control cost $\mathcal{L}_c$ is defined as follows:

$$\mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right) = \mathop{\mathbb{E}}_{\boldsymbol{z}_{[T_c]}}\left[\mathcal{L}_{T_c}\left(\boldsymbol{z}_{[T_c]}, \boldsymbol{u}_{[T_c]}\right)\right] + \sum_{t=1}^{T_c-1} \mathop{\mathbb{E}}_{\boldsymbol{z}_{[t]}}\left[\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right)\right] \tag{2.8}$$

Algorithm 1 Trajectory Optimization using DBNN models

Input: Initial state $\boldsymbol{z}_{[1]}$, control penalty $\mathcal{L}_c(\cdot)$

Output: Sequence of optimal control inputs $\left\{\boldsymbol{u}^*_{[t]}\right\}_{t=1}^{T_c}$

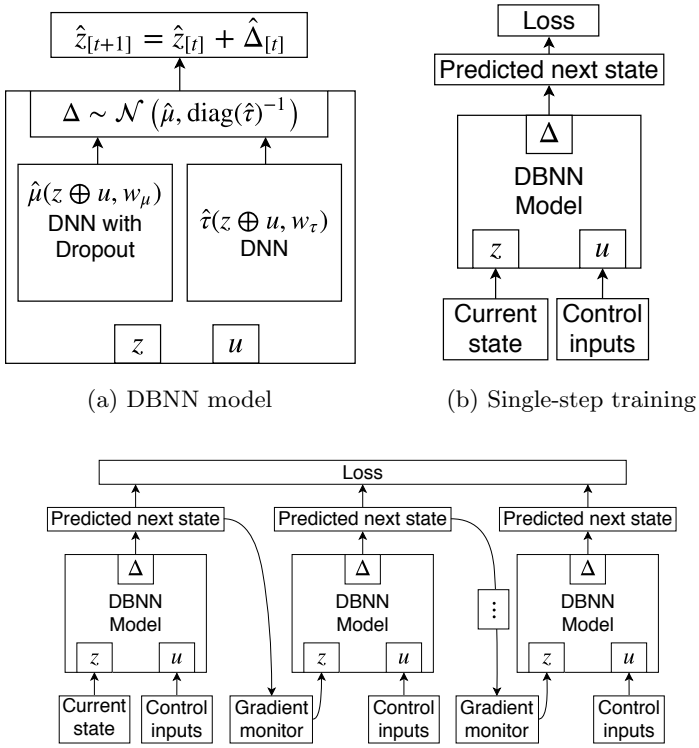
1: Initialize dataset $\mathcal{D}$ with U_o system trajectories of length T_c, using random control inputs
$\mathcal{D} \leftarrow \left\{(\boldsymbol{\Delta}^{(i)}_{[t]}, \boldsymbol{z}^{(i)}_{[t]}, \boldsymbol{u}^{(i)}_{[t]})\right\}_{i,t=1}^{i=U_o,t=T_c}$
2: **for** $j = 1$ to *max-iter* **do**
3: **Modeling:** Fit dynamics by minimizing:
$\phi^* = \arg\min \mathcal{L}(\phi)$
where $\mathcal{L}(\phi)$ is either:
- The single-step loss in Eq. (2.9) (section 2.3.1)
- The multi-step loss in Eq. (2.10) (section 2.3.2)

4: **Planning:** Run trajectory optimization by minimizing the control cost in Eq. (2.8):
$\boldsymbol{u}^*_{[t]} = \arg\min \mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right)$
Section 2.3.3 describes how the expectations on Eq. (2.8) are approximated using particles.
5: Collect U new trajectories from the real system using $\boldsymbol{u}_{[t]} = \boldsymbol{u}^*_{[t]} + \lambda\boldsymbol{\epsilon}$, where λ is used as an exploration parameter
$\mathcal{D} \leftarrow \mathcal{D} \cup \left\{(\boldsymbol{\Delta}^{(i)}_{[t]}, \boldsymbol{z}^{(i)}_{[t]}, \boldsymbol{u}^{(i)}_{[t]})\right\}_{i,t=1}^{i=U,t=T_c}$
6: **end for**
7: **return** Sequence of optimal controls $\boldsymbol{u}^*_{[t]}$

Algorithm 1 describes the open-loop trajectory optimization approach using DBNN models. Planning is performed completely offline and depends only on the control loss and the long-term estimations of the DBNN model. We used ADAM [55] for all optimization tasks. In each iteration, new data-samples are collected from the system in order to to continuously improve the learned DBNN model. Fig. 2.3b and 2.3c show the flow-chart for single-step and multi-step training, respectively. Single-step uses one-step prediction for learning the system dynamics. Multi-step uses predictions over several time steps. The following sections describe the details of single-step and multi-step approaches for fitting the dynamics.

2.3.1. Training using single-step prediction

Given the Markovian properties of fully-observable systems, a common approach to fit the dynamics is by using single-step predictions [45] [56]. Under this approach, the training loss fits the dynamics by only taking into account predictions made one step ahead. For a mini-batch $\left\{(\boldsymbol{\Delta}^{(i)}, \boldsymbol{z}^{(i)}, \boldsymbol{u}^{(i)})\right\}$ of $|\boldsymbol{X}|$



(a) DBNN model

(b) Single-step training

(c) Multi-step training

Figure 2.3.: Modeling dynamic systems using DBNNs

number of samples the loss is expressed as follows:

$$\mathcal{L}(\phi) = \frac{1}{2\,|\boldsymbol{X}|} \sum_{i=1}^{|\boldsymbol{X}|} \mathcal{L}_R \left(\boldsymbol{\Delta}^{(i)}, \boldsymbol{z}^{(i)} \oplus \boldsymbol{u}^{(i)} \right) + \frac{1}{|\mathcal{D}|} KL\left(q_\phi(w) \| p(w)\right) \tag{2.9}$$

where $|\mathcal{D}|$ is the total number of samples in the training dataset. Note that the single-step loss in Eq. (2.9) is defined w.r.t. the parameters ϕ of the variational distribution $q_\phi(w)$. As shown in Fig. 2.3a, Dropout is only applied to $\hat{\boldsymbol{\mu}}$.

2.3.2. Training using multi-step predictions

Improved performance has been observed when fitting sequential models using multi-step/long-term predictions [35]. By forcing the model to use its own predictions to predict further in the future, we ensure that the model takes into account the compounding errors that get propagated over time, thus improving generalization and encouraging learning better internal representations.

Multi-step training is performed using mini-batches of size $|\boldsymbol{X}|$, composed of an initial state $\boldsymbol{z}_{[1]}^{(i)}$ with a sequence of control and state increments:

$$\text{mini-batch} = \left\{ \boldsymbol{z}_{[1]}^{(i)}, \left\{ \left(\boldsymbol{\Delta}_{[t]}^{(i)}, \boldsymbol{u}_{[t]}^{(i)} \right) \right\}_{t=1}^{T_m} \right\}_{i=1}^{|\boldsymbol{X}|}$$

where $\boldsymbol{u}_{[t]}^{(i)}$ is the control input of sample i at time t and $\boldsymbol{\Delta}_{(t)}^{(i)}$ is the corresponding state increment.

Training using multi-step predictions is achieved using the following loss:

$$\mathcal{L}(\phi) = \frac{1}{2\left|\boldsymbol{X}\right| T_m} \sum_{i,t=1}^{|\boldsymbol{X}|,T_m} \mathcal{L}_R\left(\boldsymbol{\Delta}_{[t]}^{(i)}, \hat{\boldsymbol{z}}_{[t]}^{(i)} \oplus \boldsymbol{u}_{[t]}^{(i)}\right) + \frac{1}{|\mathcal{D}|} KL\left(q_\phi(w) \| p(w)\right) \tag{2.10}$$

where $\hat{\boldsymbol{z}}_{[t]}^{(i)}$ is the estimated state at time t:

$$\hat{\boldsymbol{z}}_{[t]}^{(i)} = \boldsymbol{z}_{[1]}^{(i)} + \sum_{k=1}^{t-1} \hat{\boldsymbol{\mu}}_{[k]}^{(i)} \odot \boldsymbol{\epsilon} \frac{1}{\sqrt{\hat{\boldsymbol{\tau}}_{[k]}^{(i)}}} \tag{2.11}$$

where:

$$\hat{\boldsymbol{\mu}}_{[k]}^{(i)} = \hat{\boldsymbol{\mu}}\left(\hat{\boldsymbol{z}}_{[k]}^{(i)} \oplus \boldsymbol{u}_{[k]}^{(i)}, w_\mu\right), \quad w_\mu \sim q_\phi(w)$$
$$\hat{\boldsymbol{\tau}}_{[k]}^{(i)} = \hat{\boldsymbol{\tau}}\left(\hat{\boldsymbol{z}}_{[k]}^{(i)} \oplus \boldsymbol{u}_{[k]}^{(i)}, w_\tau\right)$$

The loss in Eq. (2.10) is a standard unrolling of the neural-network, usually employed in the Back-propagation Through-Time algorithm [57].

The multi-step propagation in Eq. (2.11) uses a single sample realization of the DBNN probability distribution. This single sample realization is used in the multi-step loss from Eq. (2.10), which can be interpreted as a single-sample MC approximation of the expectation in Eq. (2.3). This method provides a noisy but efficient way to perform mini-batch optimization for learning the dynamics of the system. A more accurate estimation of the multi-step loss can be made by using multiple-particles (see section 2.3.3). However, using multiple particles leads to excessive computational burden. In the experiments we evidenced that using a single particle is enough to obtain a stable optimization for learning the system dynamics.

One of the challenges posed by multi-step training is unstable gradient propagation. Back-propagation is well known to suffer from exploding/vanishing gradient problems when applied over unrolled networks [58]. Furthermore, compounding errors that result from model mismatch can also destabilize the optimization.

In order to ensure a stable propagation of gradients, we took the following actions: 1) weights are initialized according to [59]; 2) impose the bounds $(\boldsymbol{\tau}_{\min}, \boldsymbol{\tau}_{\max})$ on $\hat{\tau}$; 3) monitor the norm of the gradients.

Heteroscedastic models are particularly sensitive to unstable gradient propagation when trained using multi-step predictions. The instability can be understood by looking at the gradients of $\mathcal{L}_R$ w.r.t. $\hat{\boldsymbol{\tau}}$ and $\hat{\boldsymbol{\mu}}$ (see Appendix B.2 for derivation):

$$\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\tau}}} = -\frac{1}{\hat{\boldsymbol{\tau}}} + (\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}})^2 \tag{2.12}$$
$$\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\mu}}} = -2\hat{\boldsymbol{\tau}} \odot (\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}}) \tag{2.13}$$

In order to ensure stable gradient propagation, we need to ensure the value of $\hat{\boldsymbol{\tau}}$ is neither excessively large nor small. Eq. (2.12) shows how small values of $\hat{\boldsymbol{\tau}}$ can cause gradients to explode. Eq. (2.13) shows how large values of $\hat{\boldsymbol{\tau}}$ from over-confident estimations can also lead to exploding gradients when the difference between $\boldsymbol{\Delta}$ and $\hat{\boldsymbol{\mu}}$ is large. In Eq. (2.11) we can also see how extremely small values of $\hat{\boldsymbol{\tau}}$ results in excessive errors being propagated over time, which in turn results in large gradients in both Eq. (2.12) and (2.13) (large $\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}}$).

We use the following activation function in the output layer of $\hat{\tau}$ in order to constrain the value of $\hat{\boldsymbol{\tau}}$ to be in a given range $(\boldsymbol{\tau}_{min}, \boldsymbol{\tau}_{max})$:

$$f(\boldsymbol{h}) = (\boldsymbol{\tau}_{\max} - \boldsymbol{\tau}_{\min}) \operatorname{Sig}(\boldsymbol{h}) + \boldsymbol{\tau}_{\min} \tag{2.14}$$

where $\operatorname{Sig}(\cdot)$ is the sigmoid function and $\boldsymbol{h}$ represents the pre-activations on the output layer. Because the output of the sigmoid function is in the range $(0, 1)$, the maximum value that $\hat{\boldsymbol{\tau}} = f(\boldsymbol{h})$ can take is $\boldsymbol{\tau}_{\max}$ when $\operatorname{Sig}(\boldsymbol{h}) = \mathbf{1}$. Likewise, the minimum value that $\hat{\boldsymbol{\tau}}$ can take is $\boldsymbol{\tau}_{\min}$ when $\operatorname{Sig}(\boldsymbol{h}) = \mathbf{0}$. Note that the sigmoid activation function $f(\boldsymbol{h})$ is only applied in the output layer. For the hidden layers, we used the ReLU activation function.

As a final line of defense for exploding gradients, we place gradient monitors between the unrolled networks. These monitors evaluate the norm of the gradients and reset the gradients to zero if the norm is larger than a predefined value. Fig. 2.3c shows the unrolling of the neural network with the gradient monitors for the multi-step training approach.

2.3.3. Long-term trajectory estimations using the learned DBNN model

Trajectory optimization uses the long-term estimations of the DBNN model to find the control sequence that minimizes the control cost. In this case, the state trajectory is estimated using a standard sequential Monte-Carlo approach [56]. Given an initial state $\boldsymbol{z}_{[1]}$ and the list of control inputs $\left\{\boldsymbol{u}_{[t]}\right\}_1^{T_c}$, we estimate the system state trajectory using a set of P particles, all starting from the initial $\boldsymbol{z}_{[1]}$ state. The state $\hat{\boldsymbol{z}}_{[t]}^{(p)}$ of each particle p at time t is estimated as follows:

$$\hat{\boldsymbol{z}}_{[t]}^{(p)} = \boldsymbol{z}_{[1]} + \sum_{k=1}^{t-1} \hat{\boldsymbol{\mu}}_{[k]}^{(p)} \odot \boldsymbol{\epsilon} \frac{1}{\sqrt{\hat{\boldsymbol{\tau}}_{[k]}^{(p)}}} \tag{2.15}$$

where:

$$\begin{aligned} \hat{\boldsymbol{\mu}}_{[k]}^{(p)} &= \hat{\boldsymbol{\mu}}\left(\hat{\boldsymbol{z}}_{[k]}^{(p)} \oplus \boldsymbol{u}_{[k]}, w_\mu\right), \quad w_\mu \sim q_\phi(w) \\ \hat{\boldsymbol{\tau}}_{[k]}^{(p)} &= \hat{\boldsymbol{\tau}}\left(\hat{\boldsymbol{z}}_{[k]}^{(p)} \oplus \boldsymbol{u}_{[k]}, w_\tau\right) \end{aligned}$$

Eq. (2.15) is a standard unrolling of the neural-network. The expectations on the control cost in Eq. (2.8) are approximated using the sample mean of the particles:

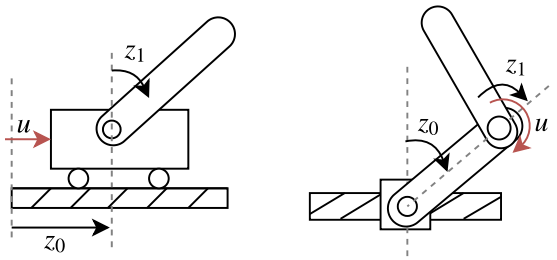
$$\mathop{\mathbb{E}}_{\boldsymbol{z}_{[t]}}\left[\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right)\right] \approx \frac{1}{P} \sum_{p=1}^{P} \mathcal{L}_t\left(\hat{\boldsymbol{z}}_{[t]}^{(p)}, \boldsymbol{u}_{[t]}\right)$$

The standard deviation of the particles is used for measuring and visualizing the uncertainty of the predicted state trajectory.

2.3.4. Computational complexity

The dominant operation in Algorithm 1 is the vector-matrix products performed during the DBNN forward and backpropagation computations. The complexity of these operations depends primarily on the size of the weight matrices. Assuming the networks $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\tau}})$ are composed by (L_μ, L_τ) hidden layers, with (M_μ, M_τ) hidden units in each layer, the number parameters of the DBNN can be roughly approximated as follows:

$$|w| = L_\mu\left(M_\mu^2\right) + L_\tau\left(M_\tau^2\right) \tag{2.16}$$



(a) Cartpole (b) Acrobot

Figure 2.4.: Illustration of the benchmark problems being considered

We use this expression to quantify the space complexity of the algorithm. The time complexity of forward and backward computations for a mini-batch of size $|\boldsymbol{X}|$ can be approximated as follows:

$$|\boldsymbol{X}|\, T_m \left(L_\mu \left(M_\mu^2\right) + L_\tau \left(M_\tau^2\right)\right) \tag{2.17}$$

where T_m is the number of multi-step predictions. Long-term trajectory computations used for trajectory optimization (control) have an equivalent time complexity of $PT_c \left(L_\mu \left(M_\mu^2\right) + L_\tau \left(M_\tau^2\right)\right)$, where P is the number of particles.

Eq. (2.16) and (2.17) demonstrate further potential benefits of using multi-step and heteroscedastic models. The dominant term in Eq. (2.16) and (2.17) is the square of hidden units M_μ^2 and M_μ^2. This shows that increasing the number of hidden units for a homoscedastic model is very expensive. On the other hand, increasing the number of time steps T_m in multi-step training has a much lower cost regarding time complexity, and no cost in space complexity is incurred. For our application, space complexity is more valuable than time complexity because the optimizations are performed offline.

Eq. (2.16) also shows that adding a heteroscedastic model with N hidden units is cheaper than adding such units into a homoscedastic model:

$$L_\mu \left(M_\mu^2\right) + L_\mu \left(N^2\right) < L_\mu \left(M_\mu + N\right)^2$$

2.4. Experiments

Underactuated mechanical systems are commonly found in industrial applications [60] and provide interesting benchmark problems for controls. Therefore, for experimental evaluation, we selected the Cartpole (Fig. 2.4a) and Acrobot (Fig. 2.4b) as benchmark problems. In addition to having underactuated dynamics, these systems also have unstable fix-points and in the case of the Acrobot, chaotic behavior when no control is applied.

The *objectives* of the experiments are: 1) validate the use of the presented stochastic modeling framework for obtaining long-term predictions with uncertainty; 2) evaluate the advantages of using heteroscedastic models; 3) evaluate the advantages of multi-step training.

The *highlights* of the experimental evaluation are: 1) training using multi-step predictions provided better generalization and faster convergence rates in the trajectory optimization task, compared to single-step training; 2) heteroscedastic models provided estimations with lower uncertainty when compared to homoscedastic models; 3) the presented methodology is suitable for real-world applications; 4) multi-step training is stable, even when we use only one particle for the Monte-Carlo estimations during training. The experiments demonstrated that the methodology is able to provide accurate long-term predictions of non-linear underactuated systems. The methodology is suitable for real-world applications as the experiments demonstrate it is sample-efficient, i.e. it requires a relatively low number of

samples from the real system. Furthermore, planning is performed completely offline. All expensive optimizations are executed offline, which means that they are not required to run in real-time.

2.4.1. Experiment setup

In our experiments, the trajectory optimization task is to perform the swing-up maneuver for Cartpole and Acrobot. The maneuver entails driving the poles from a downward position at time $t = 0$, to an upward position at $t = T_c$.

The state for the Cartpole is four-dimensional $\boldsymbol{z} = [z_0, z_1, \dot{z_0}, \dot{z_1}]$, where $z_0 \in [-1, 1]$ represents the position of the cart and $z_1 \in [-4, 4]$ represents the position of the pole in radians. $\dot{z_0}$ and $\dot{z_1}$ represent the time derivatives of z_0 and z_1, respectively.

The state for the Acrobot is also four-dimensional $\boldsymbol{z} = [z_0, z_1, \dot{z_0}, \dot{z_1}]$, where $z_0 \in [-4, 4]$ represents the position of the first pole and $z_1 \in [-4, 4]$ the position of the second pole in radians.

For both Cartpole and Acrobot, the coordinate system was configured such that the $[0, 0, 0, 0]$ state corresponds to the poles in the upward position. The simulations were performed using OpenAI Gym [61] with PyBullet [62]

The objective of the trajectory optimization is to reach the target state $\boldsymbol{z}^*_{[T_c]} = [0, 0, 0, 0]$ at time T_c. To achieve this objective, the losses $\mathcal{L}_{T_c}, \mathcal{L}_t$ from Eq. (2.8) were defined as follows:

$$\mathcal{L}_{T_c}\left(\boldsymbol{z}_{[T_c]}, \boldsymbol{u}_{[T_c]}\right) = (\boldsymbol{z}_{[T_c]} - \boldsymbol{z}^*_{[T_c]})^T \boldsymbol{Q}_T (\boldsymbol{z}_{[T_c]} - \boldsymbol{z}^*_{[T_c]}) \tag{2.18}$$

$$\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right) = \boldsymbol{u}^T_{[t]} \boldsymbol{Q}_u \boldsymbol{u}_{[t]} \tag{2.19}$$

where $\boldsymbol{Q}_T, \boldsymbol{Q}_u$ are diagonal matrices. $\mathcal{L}_{T_c}$ penalizes for reaching a state different than $\boldsymbol{z}^*_{[T_c]}$ at the end of the trajectory, while $\mathcal{L}_t$ penalizes for high control inputs.

We evaluated the performance of different DBNN architectures using two criteria: 1) The performance on the open-loop control task; 2) The quality of the DBNN long-term predictions.

Given that the control sequence $\boldsymbol{u}*$ obtained with Algorithm 1 is applied in open-loop, accurate long-term predictions are necessary to successfully reach the target state $\boldsymbol{z}^*_{[T]}$. We measured the quality of long-term trajectory predictions using: 1) the deviation of the real trajectory $\boldsymbol{z}$ from the predicted trajectory $\hat{\boldsymbol{z}}$; 2) the standard deviation of the estimations; 3) the containing-ratio of long-term predictions on a test dataset. The containing-ratio (CR-n) is the percentage of real samples inside $n-$ standard deviations.

We measured the deviation of the real trajectory ($\boldsymbol{z}_{[t]}$) from the predicted probability distribution ($\hat{\boldsymbol{z}}_{[t]}$) using the following metrics:

$$\delta_\mu(\boldsymbol{z}, \hat{\boldsymbol{z}}) = \frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{z}_{[t]} - \text{Mean}(\hat{\boldsymbol{z}}_{[t]})\|_2^2 \tag{2.20}$$

$$\delta_\sigma(\boldsymbol{z}, \hat{\boldsymbol{z}}) = \left\| \frac{1}{T} \sum_{t=1}^{T} \frac{|\boldsymbol{z}_{[t]} - \text{Mean}(\hat{\boldsymbol{z}}_{[t]})|}{\text{STD}(\hat{\boldsymbol{z}}_{[t]})} \right\| \tag{2.21}$$

where $\text{Mean}(\hat{\boldsymbol{z}}_{[t]})$ and $\text{STD}(\hat{\boldsymbol{z}}_{[t]})$ are the sample-mean and sample standard-deviation of the particles in Eq. (2.15). Eq. (2.21) provides a natural way to interpret the accuracy of the predictions in terms of a Standard Score (z-score). The goal is to obtain long-term estimations with low uncertainty. On the other hand, we do not want over-confident estimations, the real trajectory should be contained inside three standard deviations of the predicted distribution.

Table 2.1 lists the architectures considered in the experiments. OS stands for homoscedastic-single-step, ES heteroscedastic-single-step, OM homoscedastic-multi-step, and ES heteroscedastic-multi-step. The table shows the number of hidden units and layers used for $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\tau}}$. For example, $\hat{\boldsymbol{\mu}}[50, 50]$ means a DBNN with two hidden layers and 50 hidden units in each layer. For the Cartpole we used

Table 2.1.: Model Architectures

	Cartpole			Acrobot		
Architecture	μ	τ	T_m	μ	τ	T_m
LSTM	$[10, 10]$	-	30	$[25, 25]$	-	30
OS (baseline)	$[50, 50]$	-	-	$[100, 100]$	-	-
ES	$[50, 50]$	$[25]$	-	$[100, 100]$	$[50]$	-
OM	$[50, 50]$	-	10	$[100, 100]$	-	10
EM	$[50, 50]$	$[25]$	10	$[100, 100]$	$[50]$	10

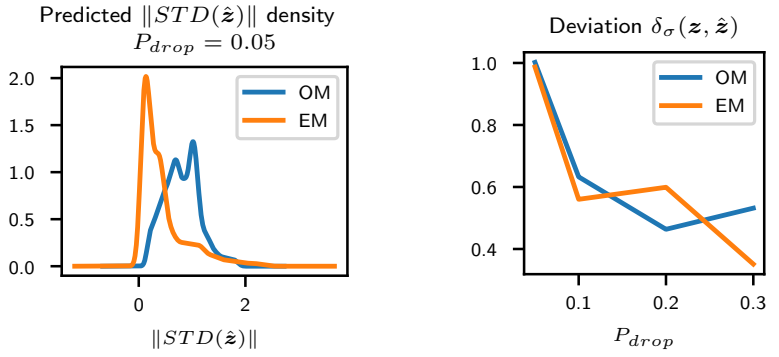
$p_{drop} = 0.1, T_c = 50$ and for the Acrobot $p_{drop} = 0.03, T_c = 110$. We used ReLU activation functions for the hidden layers.

To demonstrate the advantages of stochastic models over deterministic models, we compared the performance of the presented stochastic approach with a variation of the deterministic sequence-to-sequence LSTM model presented in [35]. Given that we consider only fully observable systems, instead of an LSTM encoder, we used a linear model to obtain the initial state of the LSTM decoder described in [35].

2.4.2. Results

Table 2.2 shows the control performance achieved using the architectures from Table 2.1. Given that the target state is located at $\boldsymbol{z}^* = \mathbf{0}$, we evaluate the control performance using: 1) the average absolute value of the final state $\left|\boldsymbol{z}^*_{T_c}\right|$; 2) the average magnitude of the final state $\|\boldsymbol{z}^*_{T_c}\|$. We also report the average control cost $\mathcal{L}_c$ of the real trajectory. Overall, Table 2.2 shows the EM model had the best performance on the control task for both Cartpole and Acrobot, with the lowest control cost $\mathcal{L}_c$. The deterministic LSTM had the worst performance in the control task for both systems.

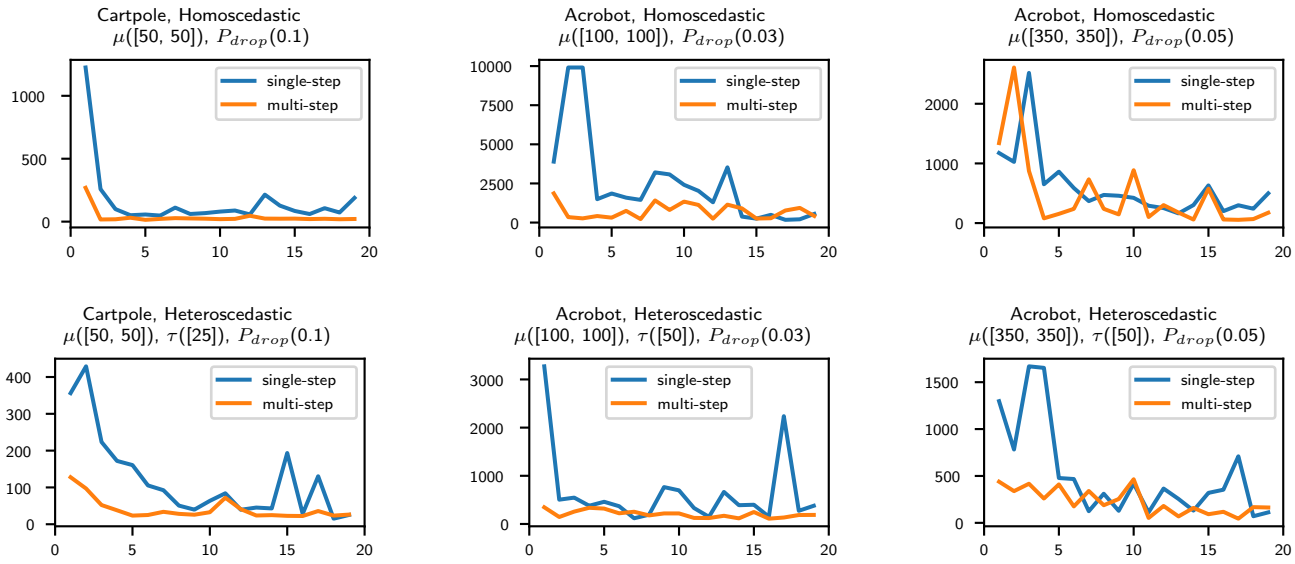
Table 2.3 shows the quality of the estimated long-term trajectories. The table shows the STD magnitude and deviation of long-term estimations on: 1) a testing dataset, composed of trajectories that were not used to train the model; 2) the optimal trajectories found through iterations of Algorithm 1. Overall, EM model provided the estimations with the lowest uncertainty. EM model also provided the



(a) Density distribution of $\|STD\|$

(b) Deviation δ_σ for different values of Dropout

Figure 2.5.: Comparison between OM and EM estimations of long-term trajectories (Cartpole) on testing dataset. a) OM provides estimations with lower uncertainty ($\|STD\|$ closer to zero) b) The deviation δ_σ is reduced by increasing dropout.



(a) Cartpole

(b) Acrobot (small sized DBNNs)

(c) Acrobot (large sized DBNNs)

Figure 2.6.: Final trajectory cost $\mathcal{L}_T$ for singe-step and multi-step training during iteration $j = [1, ..., 20]$ of Algorithm 1. Multi-step training improves the convergence rate. Optimal trajectories are found after few iterations, demonstrating the viability for real-world applications.

most accurate trajectories according to δ_μ.

The LSTM model was unable to provide accurate long-term predictions. In addition to being unable to provide uncertainty estimations, LSTM was prone to over-fitting, which explains the relative small number of units used for this model and the higher number of T_m used for training. Table 2.3 shows LSTM had the highest mean deviation σ_μ on the testing dataset.

Table 2.4 shows the containing-ratios on the testing dataset, with $T = T_c$ time-step estimations. Table 2.4 shows the containing-ratios for 1σ (CR-1), 2σ (CR-2), and 3σ (CR-3), where σ stands for standard deviation. The table also shows the skewness of the test samples after computing their corresponding z-score. This table allows us to compare the quality of the uncertainty estimations provided by the different models. Overall, we see that the containing-ratios approximate the (68-95-99.7) rule of the Normal distribution, with the EM model providing the best CRs.

2.4.3. Comparative analysis: Heteroscedastic vs Homoscedastic

Heteroscedastic models provided estimations with lower uncertainty compared to homoscedastic models. Table 2.3 shows the estimations of ES and EM models have lower standard deviations compared to OS and OM models. For the Acrobot model, the reduction is considerable, a result that can be attributed to the higher difficulty posed by the Acrobot dynamics. Fig. 2.5a shows the distribution of estimated $\|STD\|$ values on the test dataset. This figure shows in more detail the lower uncertainty provided by heteroscedastic models.

Although heteroscedastic models provided estimations with lower uncertainty, the experiments also showed that heteroscedastic models are more likely to provide overconfident estimations. Table 2.4 shows that for the Cartpole, ES had the lowest CR3 score and the worst skewness.

Fig. 2.5b shows how increasing Dropout can be used to reduce the deviation $\delta_\sigma(\boldsymbol{z}, \hat{\boldsymbol{z}})$, reducing the chance of obtaining over-confident estimations. This serves as a tool for alleviating potential model-

Table 2.2.: Control performance of Algorithm 1

	Average $\lvert \boldsymbol{z}_{[T_c]} \rvert$				Average	Average
	$\lvert z_0 \rvert$	$\lvert z_1 \rvert$	$\lvert z_2 \rvert$	$\lvert z_3 \rvert$	$\lVert \boldsymbol{z}_{[T_c]} \rVert$	$\mathcal{L}_c$
Cartpole						
LSTM	0.37	1.37	1.05	5.29	5.73	25.785
OS (baseline)	0.52	0.85	2.66	4.56	5.99	15.995
ES	0.56	0.75	1.24	4.21	4.98	15.061
OM	0.25	0.35	2.05	1.55	2.96	5.898
EM	0.41	0.26	2.75	1.04	3.07	**5.298**
Acrobot						
LSTM	2.65	2.51	3.92	6.48	10.22	197.366
OS (baseline)	1.36	3.10	2.68	1.80	5.34	81.991
ES	1.42	1.76	2.84	2.68	5.02	67.654
OM	0.99	1.18	3.34	4.65	6.50	60.519
EM	0.50	1.61	2.45	2.74	4.56	**21.025**

Table 2.3.: Long-term prediction performance on test dataset and Optimal trajectories

	Test dataset			Optimal trajectory $\boldsymbol{z}^*$		
	δ_μ	$\lVert STD \rVert$	δ_σ	δ_μ	$\lVert STD \rVert$	δ_σ
Cartpole						
LSTM	0.367	-	-	5.59	-	-
OS (baseline)	0.252	2.96	0.72	5.59	0.60	1.03
ES	0.282	0.77	0.74	3.15	0.23	1.20
OM	0.086	1.46	0.56	4.84	0.52	**0.99**
EM	**0.071**	**0.43**	**0.42**	**1.55**	**0.21**	1.01
Acrobot						
LSTM	12.71	-	-	48.74	-	-
OS (baseline)	3.70	27.91	0.85	65.53	3.26	0.67
ES	1.76	6.64	0.87	17.75	0.85	1.45
OM	1.47	13.88	**0.57**	34.69	2.28	**0.60**
EM	**0.80**	**4.15**	0.59	**6.19**	**0.68**	1.15

mismatch problems.

2.4.4. Comparative analysis: single-step vs multi-step training

Multi-step training had the advantage of improving generalization, providing accurate long-term predictions with lower deviation and better containing-ratios in the testing dataset. The combination of heteroscedastic and multi-step training provided the best performance in the control (Table 2.2) and estimation tasks (Tables 2.3 and 2.4).

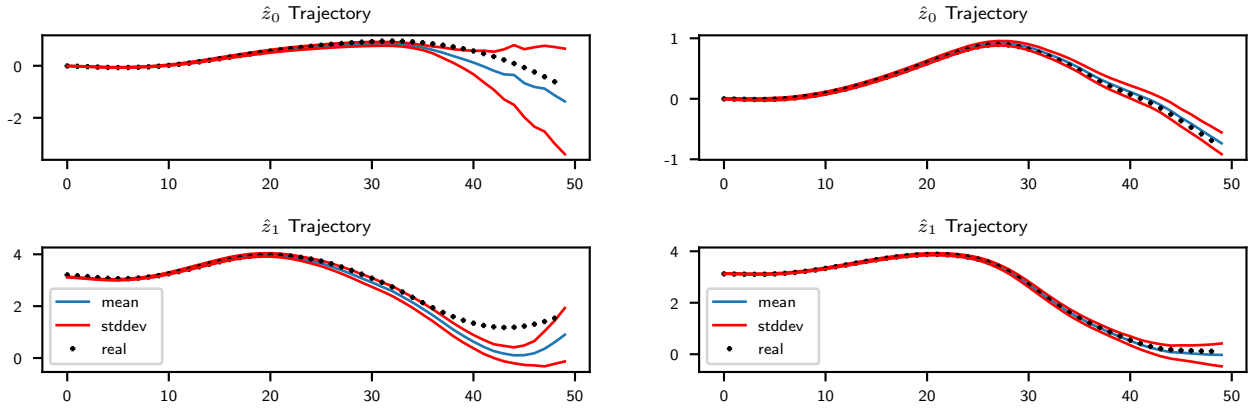
Tables 2.3 and 2.4 serve as evidence of the improved generalization of multi-step training. Table 2.3 shows that multi-step training reduces the deviation (δ_μ and δ_σ) between the predictions and the real behavior of the system. EM models provided accurate estimations with the lowest uncertainty ($\lVert STD \rVert$) without being overconfident (low deviation and best containing-ratios).

Table 2.4 shows that multi-step training improves the containing-ratios of long-term predictions. This is of particular interest when using heteroscedastic models. Multi-step training alleviates the problem of overconfident estimations given by ES models. The best containing-ratios correspond to EM models.

Fig. 2.6 shows a comparison of the performance between single-step and multi-step training using different architectures. The figure shows the value for the final control cost $\mathcal{L}_{[T_c]}$ on each iteration j of Algorithm 1. Fig. 2.6 shows that for both, Cartpole and Acrobot, multi-step training provides faster convergence rates for the trajectory optimization task.

Table 2.4.: Containing-Ratios of Long-term predictions on test dataset

	CR-1	CR-2	CR-3	skewness
Cartpole				
OS (baseline)	0.52	0.96	1.00	[-0.09 , 0.05 , -0.71 , 0.34]
ES	0.78	0.94	0.96	[-4.92 , -1.68 , -10.87, -1.26]
OM	0.67	0.97	1.00	[0.09 , -0.61 , -0.07, -0.33]
EM	0.79	1.00	1.00	[0.21 , 0.92 , 1.27, 2.35]
Acrobot				
OS (baseline)	0.35	0.91	0.99	[0.33 , 0.13 , 0.05 , 0.13]
ES	0.47	0.93	0.99	[-0.09 , 0.12 , -0.02 , -0.29]
OM	0.58	0.97	1.00	[0.81 , 0.29 , 0.29 , 0.23]
EM	0.70	0.95	0.99	[0.10 , 0.55 , 0.30 , -0.10]



(a) Estimations during first iterations have high uncertainty ($\|STD\|$)

(b) Estimations on final iterations have low uncertainty ($\|STD\|$)

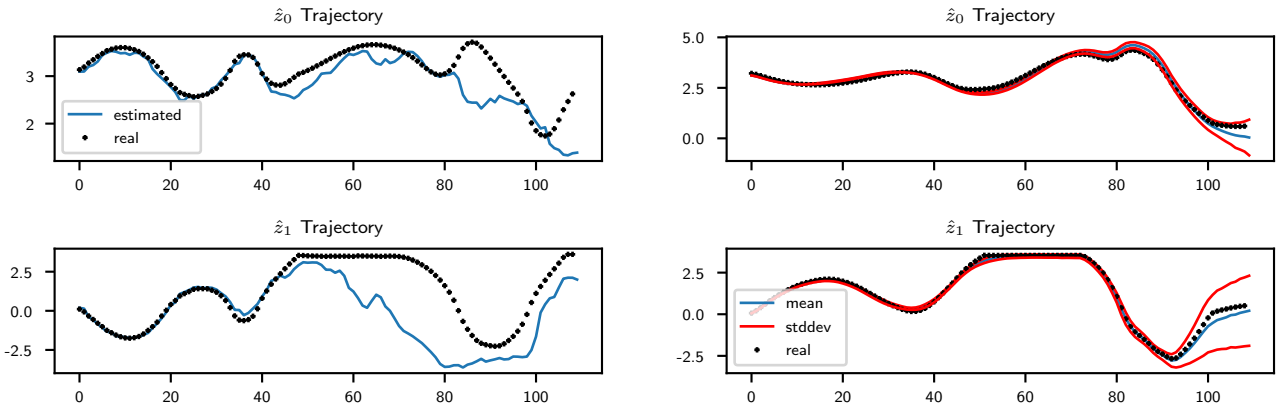
Figure 2.7.: Estimated optimal trajectories found during the execution of Algorithm 1 for the Cartpole. The standard deviation is used to quantify and visualize the uncertainty. The algorithm is able to find optimal open-loop trajectories, providing accurate long-term predictions of the system trajectory with low uncertainty.

Fig. 2.6 also shows the viability of the algorithm for real-world control applications. By iteration $j = 7$ most of the experiments had converged to an optimal trajectory. For the Cartpole (Fig. 2.6a), the algorithm converges after the first couple iterations. For the experiments, we used $U_o = 20$ initial trajectories and $U = 15$ trajectories were collected from the simulation in each iteration of Algorithm 1. Overall, the algorithm only required 125 trajectories sampled from the system to find an optimal solution.

When compared to homoscedastic models, Fig. 2.6b and 2.6c show that heteroscedastic models allowed us to use smaller DBNNs. Fig. 2.6c shows that the performance of OM models can be improved by increasing the number of hidden layers. However, this comes at the expense of higher computational complexity. Using small DBNNs has the advantage of reduced memory requirements for the optimization tasks.

2.4.5. Long-term predictions using EM models

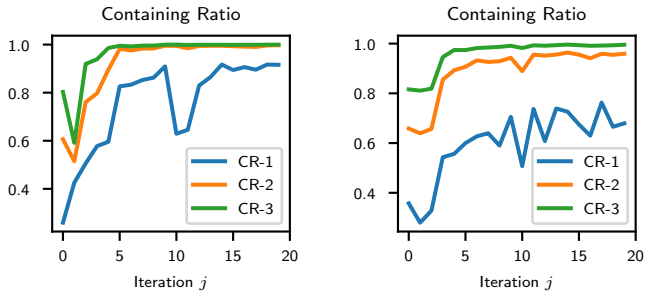
Fig. 2.7 shows the predictions of the state trajectory for the Cartpole during first (2.7a) and final (2.7b) iterations of Algorithm 1. We used the standard deviation of the predictions to visualize the uncertainty.



(a) Model-mismatch when using deterministic LSTM model. The estimated trajectory deviates from the real trajectory from t=45

(b) EM provides accurate long-term estimations that match the trajectory of the real system

Figure 2.8.: Long-term optimal trajectory estimations provided by EM and LSTM models for the Acrobot. Deterministic LSTM is unable to provide accurate long-term predictions.



(a) Cartpole

(b) Acrobot

Figure 2.9.: Containing-ratio of long-term test trajectories during iteration j of Algorithm 1. The containing-ratios approximate the (68-95-99.7) rule after few iterations.

Fig. 2.7a shows that the algorithm starts with high uncertain predictions. After convergence, Fig. 2.7b shows the trajectory estimations are made with low uncertainty.

Fig. 2.8 shows a comparison of the long-term trajectory estimations provided by LSTM and EM models. Fig. 2.8a shows that LSTM models suffer from model mismatch problems, where the estimated trajectory deviates from the real trajectory. In contrast, Fig. 2.8b shows the EM model is able to provide accurate long-term predictions.

Table 2.5 presents the best values of the final state $\boldsymbol{z}_{[T]}$ during the execution of Algorithm 1. Table 2.5 and Fig. 2.8b show that Algorithm 1 is able to find open-loop trajectories that drive the system close to the target state ($\boldsymbol{z} = 0$).

Fig. 2.9 shows the containing-ratios on the test dataset in each iteration j of Algorithm 1. Although the quality of the uncertainty estimation is poor in the first iteration, the figure shows the containing-ratios quickly approximate the (68-95-99.7) rule of the Normal distribution.

Table 2.5.: Best values of final state $\boldsymbol{z}_{[T]}$ after executing the optimal trajectory using EM model.

System	z_0	z_1[rad]	z_2	z_3[rad/s]
Cartpole	-0.40 [m]	0.03	-2.27 [m/s]	-0.47
Acrobot	-0.22 [rad]	-0.39	-1.39 [rad/s]	-0.46

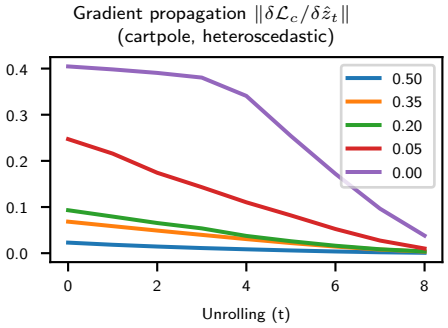


Figure 2.10.: Gradient propagation for different values of Dropout. We observe a stable increase of the gradient magnitude as the gradients are back-propagated

2.4.6. Gradient propagation

Unstable gradient propagation poses a challenge for multi-step training. Appropriate selection of ($\boldsymbol{\tau}_{min}$, $\boldsymbol{\tau}_{max}$) and weight initialization allowed us to achieve stable optimization. The bounds ($\boldsymbol{\tau}_{min}, \boldsymbol{\tau}_{max}$) were necessary to stabilize multi-step training and considerably improved the performance of the models trained using single-step approach.

Fig. 2.10 shows the magnitude of the backpropagated gradients for different dropout probabilities. The gradients grow steadily as they are back-propagated. Increasing the dropout probability reduced the rate in which gradients grow. The figure shows how gradients grow quickly without dropout, reaching the point where the gradient monitors are activated, preventing excessively large gradients from destabilizing the training.

2.5. Related work

Given the success of Deep Learning in high dimensional problems[7] and reinforcement learning [63] [64], there is an increasing interest in applying Deep Neural Networks (DNNs) in industrial applications. Forecasting [65] [35], fault diagnosis [66] [67] [68], and continuous low-level control [69] [70] [50] are some of the recent applications of DNNs.

There is also a growing interest in learning stochastic models for reinforcement learning and optimal control. In [71] Variational Bayes is used for robust identification of industrial processes. In [45] the PILCO algorithm is introduced. The algorithm uses Gaussian processes to model system dynamics and provide uncertainty estimations. In [72] Gaussian process state-space models are trained using variational inference, providing a mechanism to trade off model capacity and computation time. In [56] the PILCO algorithm is modified to use DBNNs, alleviating the problems when working with high-dimensional large datasets. In [73], a combination of bootstrapping and dropout is used to estimate uncertainties on collision avoidance tasks. In [74] the standard Gaussian process model is extended to handle sequential data by using an LSTM model. Recent work has been focused on extending the modeling capabilities of DBNN architectures to include heteroscedastic and multi-modal distributions

[75] [76].

Most control applications of ANNs and DBNNs found in literature focus on closed loop-control with some variation of actor-critic design, model predictive control (MPC) or back-propagation of the dynamics through a feedback controller. The presented approach is based on the methodology presented in [56] where DBNN are used for feedback control. In contrast to the methodologies presented by [45] [56] [75] which focus on feedback control, in this chapter we evaluated the viability of using DBNNs for open-loop trajectory optimization. In contrast to MPC approaches, our approach executes all optimizations offline. Once the optimal trajectory is computed, our approach executes the trajectory in open-loop, as opposed to MPC where the optimization is executed in real-time.

2.6. Discussion

In this chapter, we presented a methodology to improve reliability of Neural Networks in optimal planning tasks. The focus of the presented approach was to study the performance of DBNN models on providing accurate long-term estimations for open-loop planning. The accuracy of the learned stochastic model allowed us to plan completely offline and execute the trajectory in open-loop. The presented task was specifically chosen as the success of the open-loop planning task depends entirely on reliable estimations of the model. Without reliable estimations, the optimal controls obtained from the model produce completely different outcomes in the system. By modeling uncertainty, we are able to factor the approximation errors inherent from ML models in the predictions of the model,

However, applying the trajectory in open-loop has two challenges: 1) performance degrades for excessively long trajectories; 2) control is sensitive to external disturbances. As shown in Fig. 2.7b and 2.8b, as time increases, the uncertainty is also increasing. This behavior is a design choice that results from propagating the uncertainty overtime in Eq. (2.6). Due to process noise and approximation errors, the uncertainty should increase over time. However, this results in estimations with low confidence when planning over excessively long trajectories. In order to further improve control performance and guarantee robustness against external noise, close-loop control techniques such as trajectory stabilization can be used on top of our approach. Techniques such as Time-Varying LQR feedback stabilization [43] can be introduced to stabilize the system around the optimal open-loop trajectory found with Algorithm 1.

In this chapter, we presented an approach for modeling and planning under uncertainty using Deep Bayesian Neural-Networks. We presented a method for learning dynamics using multi-step predictions. The approach includes different tools for ensuring stable learning of the dynamics for heteroscedastic models. The learned model was successfully used in a trajectory optimization task. The presented data-driven modeling and planning approach was able to find optimal trajectories that can perform the swing-up maneuver for both Cartpole and Acrobot, without the need for expert knowledge of the dynamics. The learned stochastic model was able to accurately estimate the long-term state trajectories together with the uncertainty of the predictions. Compared to single-step training, multi-step training showed improved generalization and faster convergence rates in the trajectory optimization task. The accuracy of the estimated trajectory probability distributions allowed us to plan completely offline and execute the optimal trajectory in open-loop.

3. Improving interpretability of ML for physical systems by embedding domain-knowledge

This chapter is in support of contribution 2: *The Explicit Variational Gaussian Process model, a stochastic data-driven model enhanced with domain-knowledge from physics.*

- *Improving interpretability and reliability while using less amount of data when compared with fully data-driven approaches.*
- *Improving accuracy using simple physics-based priors, demonstrating that accuracy improves even when using heavily simplified physics as domain-knowledge.*
- *Scalability to large datasets using the sparse variational framework.*
- *Improving interpretability using the posterior distribution of trainable parameters, i.e. the value of the model parameters after training.*

Paper:

-

Papers that preceded this work:

- D. Marino, K. Amarasinghe, M. Anderson, N. Yancey, Q. Nguyen, K. Kenney, M. Manic, "Data-driven decision support for reliable biomass feedstock preprocessing" , IEEE Resilience Week (RWS) 2017, Wilmington, DE, USA, Sep, 18-22, 2017
- D. Marino, M. Anderson, K. Kenney, and M. Manic, "Interpretable data-driven modeling in biomass preprocessing," in Proc. 11th International Conference on Human System Interaction, IEEE HSI 2018, Gdansk, Poland, July 04-06, 2018.

Centuries of development in natural sciences and mathematical modeling provide valuable domain expert knowledge that has yet to be explored for the development of machine learning models. When modeling complex physical systems, both domain knowledge and data provide valuable information about the system. In this chapter, we present a data-driven model that takes advantage of partial domain knowledge in order to improve generalization and interpretability. The presented approach, which we call EVGP (Explicit Variational Gaussian Process), has the following advantages: 1) using available domain knowledge to improve the assumptions (inductive bias) of the model, 2) scalability to large datasets, 3) improved interpretability. We show how the EVGP model can be used to learn system dynamics using basic Newtonian mechanics as prior knowledge. We demonstrate how the addition of prior domain-knowledge to data-driven models outperforms purely data-driven models [77].

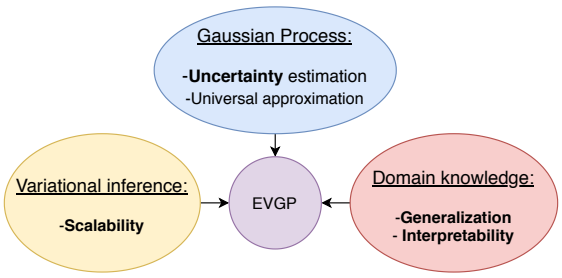


Figure 3.1.: EVGP: Variational Gaussian-Process with explicit features

3.1. Introduction

For centuries, scientists and engineers have worked on creating mathematical abstractions of real world systems. This principled modeling approach provides a powerful toolbox to derive white-box models that we can use to understand and analyze physical systems. However, as the complexity of physical systems grow, deriving detailed principled models becomes an expensive and tedious task that requires highly experienced scientists and engineers. Moreover, incorrect assumptions leads to inaccurate models that are unable to represent the real system.

Data-driven black-box models provide an appealing alternative modeling approach that requires little to none domain knowledge. These models are fit to data extracted from the real system, minimizing the problems derived from incorrect assumptions. However, using data-driven models while completely ignoring domain knowledge may lead to models that do not generalize well and are hard to understand. Completely black-box approaches ignore the structure of the problem, wasting resources [78] and making the model less explainable [79].

Gray-box models combine domain knowledge and data as both provide important and complementary information about the system. Domain knowledge can be used to construct a set of basic assumptions about the system, giving the data-driven model a baseline to build upon. Data can be used to fill the gaps in knowledge and model complex relations that were not considered by the domain expert.

In this chapter, we explore an approach for embedding domain knowledge into a data-driven model in order to improve generalization and interpretability. The presented gray-box model, which we called EVGP (Explicit Variational Gaussian Process), is a scalable approximation of a sparse Gaussian Process (GP) that uses domain knowledge to define the prior distribution of the GP. In this chapter, domain knowledge is extracted from physics-based knowledge, however the EVGP can be applied to any domain.

The work on this chapter has three cornerstones (Fig. 3.1): 1) Gaussian processes are used for learning complex non-linear behavior from data while providing uncertainty estimations, 2) Partial domain knowledge is used as prior in order to improve inductive bias, 3) Variational Inference provides advantageous scalability to large datasets. Inductive bias refers to the assumptions made by the model when doing predictions on novel data. The presented approach provides uncertainty estimations which are fundamental in order to avoid the risk associated with overconfidence in unexplored areas [72] and warns the user of possible incorrect estimations [37].

The work in this chapter is highly applicable when: 1) modeling physical systems with uncertainty estimations, 2) partial domain knowledge of the system is available, 3) large quantities of data are available. The aim is to help the engineer and take advantage of available knowledge without requiring the derivation of complex and detailed models. Instead, an engineer only has to provide simple, partially formed, models and the EVGP takes care of filling the gaps in knowledge. We show how the EVGP model can be used to learn system dynamics using basic physics laws as prior knowledge. We specifically demonstrate the approach on multi-body physics systems. We chose the pendulum as the base for our discussion to demonstrate how concepts from a simple system can be used as building blocks for priors

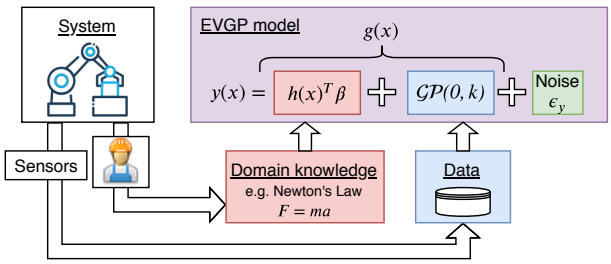


Figure 3.2.: Illustration of the EVGP model. The models uses a function $h(x)^T\beta$ to embed domain knowledge, while the GP is used to increase the model capacity to learn complex non-linear relations from data.

of 2 DOF systems up to a 7 DOF robot arm.

Our objective is to evaluate the benefits of including partially defined (imperfect) models as domain-knowledge to data-driven models. Hence, our experiments compare the performance of the EVGP approach with respect to fully data-driven (black-box) models to evaluate the benefits of the proposed approach.

The rest of the chapter is organized as follows: section 3.2 presents the EVGP approach; section 3.3 presents a set of priors derived from simplified Newtonian dynamics for the EVGP model; section 3.4 presents the experimental section where we compare the EVGP with fully data-driven models; section 3.5 presents the related work; section 3.6 concludes the chapter.

3.2. EVGP approach - Explicit Variational GP

The novel EVGP approach presented in this chapter is designed to solve regression problems under uncertainty. Figure 3.2 offers a visual representation of the approach.

Given a dataset $\mathcal{D} = (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ composed of input/output pairs of samples $\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)$, we would like to obtain a predictive distribution $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D})$ that estimates the value of the output $\boldsymbol{y}$ for a given input $\boldsymbol{x}$. The EVGP model approximates $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D})$ using variational Inference. The EVGP is defined as a distribution $p(\boldsymbol{y}|\boldsymbol{x}, w)$ where w are a set of parameters with prior distribution $p(w)$.

In the following sections we describe in detail: *A)* the EVGP model approach, *B)* the variational loss function used to train the model, *C)* the predictive distribution that approximates $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D})$. Appendix C.1 provides a quick description of acronyms and symbols for a quick reference for the reader.

3.2.1. EVGP Model Definition

The EVGP model takes the following form:

$$y = g(\boldsymbol{x}) + \epsilon_y; \quad g\left(\boldsymbol{x}\right) = h\left(\boldsymbol{x}\right)^T\boldsymbol{\beta} + f\left(\boldsymbol{x}\right) \tag{3.1}$$

where $f\left(\boldsymbol{x}\right) \sim \mathcal{GP}\left(0, k\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right)$ is a Gaussian process with kernel k, $\epsilon_y \sim \mathcal{N}\left(0, \boldsymbol{\Sigma}_y\right)$ is the observation noise and $g(\boldsymbol{x})$ is the denoised prediction for the input $\boldsymbol{x}$. Figure 3.2 offers a visual representation of the model. The following is the description of the main components of the EVGP model:

- Domain knowledge is embedded in the explicit function $h\left(\boldsymbol{x}\right)^T\boldsymbol{\beta}$, parameterized by $\boldsymbol{\beta}$. The function $h\left(\boldsymbol{x}\right)$ describes a set of features which are explicitly stated by the domain expert (hence the name of our method). $\boldsymbol{\beta}$ is modeled using a normal distribution with a prior that is also extracted from domain knowledge. In this work, $h(\boldsymbol{x})^T\boldsymbol{\beta}$ is derived from partially defined Newtonian mechanics.

- The Gaussian Process $f(\boldsymbol{x})$ adds the ability to learn complex non-linear relations, increasing the model capacity of the entire model by supplementing the function $h\left(\boldsymbol{x}\right)^T\boldsymbol{\beta}$.

Given a dataset $\mathcal{D}$, the exact predictive distribution $p(y|x, \mathcal{D})$ for the model in Eq. (3.1) is described in [80]. For the rest of the chapter, we refer to the exact distribution as EGP. Computing the EGP predictive distribution has a large computational cost and does not scale well for large datasets.

To alleviate this problem, sparse approximation methods [81] use a small set of m inducing points represented by a tuple $(\boldsymbol{f}_m, \boldsymbol{X}_m)$, where $\boldsymbol{f}_m$ is a vector of size m, and $\boldsymbol{X}_m$ is a matrix of size $m \times m$. The inducing points $(\boldsymbol{f}_m, \boldsymbol{X}_m)$ are treated as trainable parameters, which allows to approximate the predictive distribution instead of requiring the entire dataset.

In order to construct a sparse approximation for the model in Eq. (3.1), we use a set of m inducing points $(\boldsymbol{f}_m, \boldsymbol{X}_m)$ as parameters that will be learned from data. Given $(\boldsymbol{f}_m, \boldsymbol{X}_m)$ and a set of test points $(\boldsymbol{g}, \boldsymbol{X})$, the prior distribution of the model can be expressed as follows:

$$\begin{bmatrix} \boldsymbol{g}_m \mid \boldsymbol{\beta} \\ \boldsymbol{g} \mid \boldsymbol{\beta} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{H}_m \boldsymbol{\beta} \\ \boldsymbol{H}_x, \boldsymbol{\beta} \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{mm} & \boldsymbol{K}_{m\boldsymbol{x}} \\ \boldsymbol{K}_{\boldsymbol{x}m} & \boldsymbol{K}_{\boldsymbol{xx}} \end{bmatrix} \right);$$

$$\begin{aligned} \boldsymbol{H}_m &= h\left(\boldsymbol{X}_m\right); \quad \boldsymbol{H}_x = h\left(\boldsymbol{X}\right) \\ \boldsymbol{g}_m &= h(\boldsymbol{X}_m)\boldsymbol{\beta} + \boldsymbol{f}_m \end{aligned}$$

where $\boldsymbol{X}$ denotes the data matrix, where each row represents an individual sample. The rows of $\boldsymbol{H}_x$ represent the value of the function $h()$ applied to the real samples $\boldsymbol{X}$. The rows of $\boldsymbol{H}_m$ represent the value of the function $h()$ applied to the inducing (learned) points $\boldsymbol{X}_m$. Using the conditional rule for multivariate Gaussian distributions, we obtain the definition of the denoised sparse EVGP model:

$$\begin{aligned} p\left(\boldsymbol{g} \mid \boldsymbol{X}, \omega\right) \sim & \mathcal{N}\left(\boldsymbol{H}_x \boldsymbol{\beta} + \boldsymbol{\mu}_{\boldsymbol{f}|\omega}, \boldsymbol{\Sigma}_{\boldsymbol{f}|\omega}\right) \\ \boldsymbol{\mu}_{\boldsymbol{f}|\omega} = & \boldsymbol{K}_{\boldsymbol{x}m} \boldsymbol{K}_{mm}^{-1} \boldsymbol{f}_m \\ \boldsymbol{\Sigma}_{\boldsymbol{f}|\omega} = & \boldsymbol{K}_{\boldsymbol{xx}} - \boldsymbol{K}_{\boldsymbol{x}m} \boldsymbol{K}_{mm}^{-1} \boldsymbol{K}_{m\boldsymbol{x}} \end{aligned} \tag{3.2}$$

where $\omega = \{\boldsymbol{f}_m, \boldsymbol{\beta}\}$ are the parameters of our model. Here, $\boldsymbol{K}_{\boldsymbol{x}m}$, $\boldsymbol{K}_{mm}$, $\boldsymbol{K}_{m\boldsymbol{x}}$, and $\boldsymbol{K}_{\boldsymbol{xx}}$ are matrices that represent the value of the Gaussian Process kernel $k(\cdot, \cdot)$ evaluated between: $(\boldsymbol{X}, \boldsymbol{X}_m)$, $(\boldsymbol{X}_m, \boldsymbol{X}_m)$, $(\boldsymbol{X}_m, \boldsymbol{X})$, $(\boldsymbol{X}, \boldsymbol{X})$, respectively. Equation (3.2) defines our scalable EVGP model. In following sections we give prior distributions to the parameters ω and perform approximate Bayesian inference. Note that Eq. (3.2) is also conditioned on $\boldsymbol{X}_m$, however we do not indicate this explicitly in order to improve readability.

3.2.2. Variational Loss

In this section we present the loss function that we use to fit our model. In this work, we follow a variational Bayesian approach (see Appendix C.2 for a brief overview). Given a training dataset $\mathcal{D}$ and a prior distribution $p(\omega)$, we wish to approximate the posterior distribution $p(\omega|\mathcal{D})$. The posterior of ω is approximated using a variational distribution $p_\phi(w) \approx p(\omega|\mathcal{D})$ parameterized by ϕ. For the EVGP parameters $\omega = \{\boldsymbol{f}_m, \boldsymbol{\beta}\}$, we use the following variational posterior distributions:

$$p_\phi\left(\boldsymbol{f}_m\right) = \mathcal{N}\left(\boldsymbol{f}_m | \boldsymbol{a}, \boldsymbol{A}\right); \quad p_\phi\left(\boldsymbol{\beta}\right) = \mathcal{N}\left(\boldsymbol{\beta} | \boldsymbol{b}, \boldsymbol{B}\right)$$

where $(\boldsymbol{a}, \boldsymbol{b})$ are the mean values of the respective normal distributions while $(\boldsymbol{A}, \boldsymbol{B})$ are the covariance matrices. The prior-distributions for ω are also defined as multivariate normal distributions:

$$p\left(\boldsymbol{f}_m\right) = \mathcal{N}\left(\boldsymbol{f}_m \mid 0, \boldsymbol{K}_{mm}\right); \quad p\left(\boldsymbol{\beta}\right) = \mathcal{N}\left(\boldsymbol{\beta} \mid \boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta\right)$$

these prior distributions represent our prior knowledge, i.e. our knowledge before looking at the data. The matrix $\boldsymbol{K}_{mm}$ is computed using the GP kernel on the inducing points $\boldsymbol{X}_m$, while $\boldsymbol{\mu}_\beta$ and $\boldsymbol{\Sigma}_\beta$ are provided by the expert.

Given the training dataset $\mathcal{D} = (\boldsymbol{y}, \boldsymbol{X})$, the parameters ϕ of $p_\phi(\omega)$ are learned by minimizing the negative Evidence Lower Bound (ELBO). For the EVGP, the negative ELBO takes the following form:

$$\begin{aligned}\mathcal{L}(\phi) = &- \log \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{H}_x \boldsymbol{b} + \boldsymbol{K}_{xm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{a}, \boldsymbol{\Sigma}_y\right) \\ &+ \frac{1}{2}\left[\mathrm{Tr}\left(\boldsymbol{M}_1\boldsymbol{A}\right) + \mathrm{Tr}\left(\boldsymbol{M}_2\boldsymbol{B}\right) + \mathrm{Tr}\left(\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{\boldsymbol{f}|\omega}\right)\right] \\ &+ \mathcal{L}_{KL}\end{aligned} \tag{3.3}$$

where $\boldsymbol{M}_1 = \left(\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mx}\right)\boldsymbol{\Sigma}_y^{-1}\left(\boldsymbol{K}_{xm}\boldsymbol{K}_{mm}^{-1}\right)$, and $\boldsymbol{M}_2 = \boldsymbol{H}_x^T\boldsymbol{\Sigma}_y^{-1}\boldsymbol{H}_x$. The term $\mathcal{L}_{KL}$ is the KL-divergence between the posterior and prior distributions for the parameters:

$$\begin{aligned}\mathcal{L}_{KL} = &D_{KL}\left(\mathcal{N}(\boldsymbol{a}, \boldsymbol{A}) \,||\, \mathcal{N}(0, \boldsymbol{K}_{mm})\right) \\ &+ D_{KL}\left(\mathcal{N}(\boldsymbol{b}, \boldsymbol{B}) \,||\, \mathcal{N}\left(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta\right)\right)\end{aligned}$$

A detailed derivation of the variational loss in Eq. (3.3) is presented in Appendix C.3. The negative ELBO (Eq. 3.3) serves as our loss function to learn the parameters ϕ given the training dataset $\boldsymbol{y}, \boldsymbol{X}$. In order to scale to very large datasets, the ELBO is optimized using mini-batches (see Appendix C.4). In our case, the parameters of the variational approximation are: $\phi = \boldsymbol{a}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{B}, \boldsymbol{X}_m$.

3.2.3. Predictive distribution

After learning the parameters ϕ, we would like to provide estimations using the approximated variational distribution $p_\phi(\omega)$. Given a set of test points $\hat{\boldsymbol{X}}$, the estimated denoised predictive distribution is computed as an expectation of Eq. (3.2) w.r.t. $p_\phi(\omega)$:

$$\begin{aligned}p_\phi\left(\hat{\boldsymbol{g}} \mid \hat{\boldsymbol{X}}\right) = \mathop{\mathbb{E}}_{p_\phi(\omega)}\left[p\left(\hat{\boldsymbol{g}} \mid \hat{\boldsymbol{X}}, \omega\right)\right] &= \mathcal{N}\left(\hat{\boldsymbol{g}} \middle| \boldsymbol{\mu}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}}, \boldsymbol{\Sigma}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}}\right) \\ \boldsymbol{\mu}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}} &= \boldsymbol{H}_x\boldsymbol{b} + \boldsymbol{K}_{\hat{x}m}\boldsymbol{K}_{mm}^{-1}\boldsymbol{a} \\ \boldsymbol{\Sigma}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}} &= \boldsymbol{\Sigma}_{\boldsymbol{f}|\omega} + \boldsymbol{H}_{\hat{x}}\boldsymbol{B}\boldsymbol{H}_{\hat{x}}^T + \boldsymbol{K}_{\hat{x}m}\boldsymbol{K}_{mm}^{-1}\boldsymbol{A}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{m\hat{x}}\end{aligned} \tag{3.4}$$

Note that $\hat{\boldsymbol{g}}$ is just a denoised version of $\hat{\boldsymbol{y}}$. Eq. (3.4) approximates $p(\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}, \mathcal{D})$, using the learned distribution $p_\phi(\omega)$ (see Appendix C.3). The result is equivalent to [82] with the addition of $\boldsymbol{H}_x\boldsymbol{b}$ for the mean and $\boldsymbol{H}_{\hat{x}}\boldsymbol{B}\boldsymbol{H}_{\hat{x}}^T$ for the covariance. These additional terms include the information provided by the prior function $\boldsymbol{H}_x$ with the parameters $\boldsymbol{b}$ and $\boldsymbol{B}$ that were learned from data.

The approximated predictive distribution with observation noise is the following:

$$p_\phi\left(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{X}}\right) = \mathcal{N}\left(\hat{\boldsymbol{g}} \middle| \boldsymbol{\mu}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}}, \boldsymbol{\Sigma}_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}} + \boldsymbol{\Sigma}_y\right)$$

where $p_\phi\left(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{X}}\right) \approx p(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{x}}, \mathcal{D})$. In the next section, we show how Eq. (3.4) can be used to model system dynamics and predict the next state of a physical system given the control input and current state.

3.3. Embedding physics-based knowledge

In this work, we apply the EVGP model to learn the dynamics of a physical system. The state $\boldsymbol{z}_{[t]}$ of the physical system can be modeled using the following recurrent version of the EVGP:

$$\begin{aligned}\boldsymbol{z}_{[t+1]} &\sim g(\boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]}) \\ \boldsymbol{y}_{[t]} &\sim \boldsymbol{z}_{[t]} + \epsilon_y\end{aligned} \tag{3.5}$$

where $\boldsymbol{u}_{[t]}$ is the control input and $\boldsymbol{y}_{[t]}$ is the measured output of the system at time t. The symbol $\oplus$ denotes concatenation and $\boldsymbol{x}_{[t]} = \boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]}$ is the input to the EVGP model. For example, in the case of a mechatronic system: $\boldsymbol{u}_{[t]}$ are the forces applied by the actuators (e.g. electric motors); $\boldsymbol{z}_{[t]}$ is the position and velocity of the joints; $\boldsymbol{y}_{[t]}$ is the output from the sensors.

We assume independent EVGP models for each output $\boldsymbol{y}_{[t]}$ in the equation (3.5). The function $g()$ in Eq. (3.5) is modeled using the EVGP model from Eq. (3.1) and Eq. (3.4). The recurrent EVGP primarily handles model uncertainty thanks to the variational approximation used in the EVGP. We do not consider observation or process noise. [1] In the following sections we present how we can use simple Newtonian mechanics to define useful priors $h(\boldsymbol{x})^T\boldsymbol{\beta}$ for the EVGP model.

3.3.1. Priors derived from simple Newtonian dynamics

Figure 3.3a shows a simple example of a single rigid-body link. The simplest model that we can use for this system comes from Newton's second law $u = J\ddot{q_1}$, where u is the torque applied to the system, J is the moment of inertia, and q_1 is the angle of the pendulum. Using Euler discretization method, we obtain the following state-space representation that serves as the prior $h(\boldsymbol{x})^T\boldsymbol{\beta}$ for our EVGP model:

$$\begin{bmatrix} q_1[t+1] \\ \dot{q_1}[t+1] \end{bmatrix} = \begin{bmatrix} q_1[t] \\ \dot{q_1}[t] \end{bmatrix} + \Delta t \begin{bmatrix} \dot{q_1}[t] \\ \frac{1}{J}u[t] \end{bmatrix} \tag{3.6}$$

$$= \underbrace{\begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t/J \end{bmatrix}}_{\boldsymbol{\beta} \text{ prior mean } \boldsymbol{\mu}_\beta} \underbrace{\begin{bmatrix} q_1[t] \\ \dot{q_1}[t] \\ u[t] \end{bmatrix}}_{h(\boldsymbol{x}_{[t]})} \tag{3.7}$$

We refer to this prior as IF (inertia+force). Δt is the discretization time and $\begin{bmatrix} q_1[t] & \dot{q_1}[t] \end{bmatrix}^T$ is the state $\boldsymbol{z}_{[t]}$ of the system. The IF prior in Eq. (3.7) does not include gravitational effects. Gravity pulls the link to the downward position with a force proportional to $\sin q_1$. Hence, a prior that considers gravitational forces can be constructed by including $\sin(q_1[t])$:

$$\text{IFG}_{[t+1]} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & \Delta t/J & -\gamma \end{bmatrix} \begin{bmatrix} q_1[t] \\ \dot{q_1}[t] \\ u[t] \\ \sin(q_1[t]) \end{bmatrix} \tag{3.8}$$

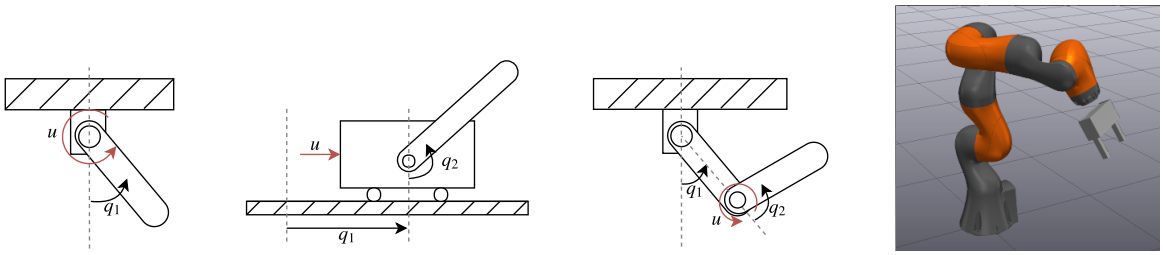
we call this prior IFG (inertia+force+gravity). We purposely did not define J and γ. One of the advantages of the presented approach is that the algorithm can learn the parameters from data if they are not available. If the user does not know the value of J and γ, a prior with large standard deviation can be provided for these parameters (large $\Sigma_{\boldsymbol{\beta}}$). Although parameters like J and γ are easy to compute for a simple pendulum, for more complex systems they may be hard and tedious to obtain. Our objective is to take advantage of domain knowledge and allow the model to fill in the gaps in knowledge.

For the rest of the chapter, priors derived from Eq. (3.7) are referred as IF priors, while Eq. (3.8) priors are referred as IFG. In the experiments (section 3.4) we compare the performance for both priors in order to illustrate how performance can be progressively improved with more detailed priors.

3.3.2. Simplified priors for Acrobot, Cartpole and IIWA

In addition to the pendulum, we consider the Acrobot, Cartpole, and IIWA systems in our analysis. For these systems, we consider much simpler priors than the exact dynamic models derived from multi-body

[1] Although observation noise can be captured by ϵ_y, the inputs to the EVGP would need to be filtered in order to properly handle observation noise (see section 3.6).



(a) Pendulum (b) Cartpole (c) Acrobot (d) IIWA [83]

Figure 3.3.: Diagrams of physical systems considered for analysis and experimentation. We show how simple priors extracted from the physics of the pendulum can be extended for systems with 2 and 7 degrees of freedom.

dynamics. We use the same principles shown in the previous section in order to get simple priors for the systems. These rules are summarized as follows:

- The position should increase proportional to the velocity by a factor of Δt.
- The position should stay the same if the velocity is zero.
- The velocity should stay the same if no external forces are applied.
- For the IFG prior, gravity pulls the links to the downward position proportional to the sine of the angle w.r.t. the horizontal plane. Gravity has no effect when the links are completely down/up.

The objective with these priors is to demonstrate how extremely simplified priors extracted with simple physics can be used to improve performance of data-driven models. Figure 3.3c shows a diagram of the Acrobot system. A simple prior for this system can be constructed using the prior in Eq. (3.7) for each one of the links of the Acrobot:

$$\text{IF} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \gamma_1 \end{bmatrix} \begin{bmatrix} q_1[t] \\ q_2[t] \\ \dot{q}_1[t] \\ \dot{q}_2[t] \\ u[t] \end{bmatrix} \tag{3.9}$$

$$\text{IFG} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -\gamma_2 & -\gamma_3 \\ 0 & 0 & 0 & 1 & \gamma_1 & 0 & -\gamma_4 \end{bmatrix} \begin{bmatrix} q_1[t] \\ q_2[t] \\ \dot{q}_1[t] \\ \dot{q}_2[t] \\ u[t] \\ \sin_1 \\ \sin_{12} \end{bmatrix} \tag{3.10}$$

where $\sin_1 = \sin(q_1[t])$, and $\sin_{12} = \sin(q_1[t] + q_2[t])$. In this case, the input $u[t]$ only drives the second

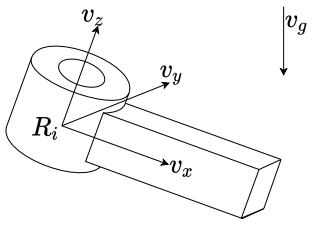


Figure 3.4.: IFG prior for a Link in 3D

link. The IF and IFG priors for the Cartpole (Figure 3.3b) are the following:

$$\text{IF} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & \gamma_1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} q_1[t] \\ q_2[t] \\ \dot{q_1}[t] \\ \dot{q_2}[t] \\ u[t] \end{bmatrix}$$

$$\text{IFG} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & \gamma_1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\gamma_2 \end{bmatrix} \begin{bmatrix} q_1[t] \\ q_2[t] \\ \dot{q_1}[t] \\ \dot{q_2}[t] \\ u[t] \\ \sin(q_2[t]) \end{bmatrix}$$

For the IIWA system, we constructed the priors following the same rules as before, with an approach that closely resembles to that used for the Acrobot. The priors were constructed using the prior in Eq. (3.7) for each one of the IIWA links. This results in a matrix $\boldsymbol{\beta}$ with a similar diagonal structure than the matrices in Eq. (3.10). To compute the IFG factors for $h()$, we derived an equation that can be used for a general 3D link (see Fig. 3.4). The equation uses forward kinematics to evaluate the contribution of the torque applied by gravity ($\boldsymbol{v}_g$):

$$h^i_{IFG} \triangleq \left(\boldsymbol{v}_x \times \boldsymbol{R}_i^T \boldsymbol{v}_g\right) \cdot \boldsymbol{v}_z$$

where $\times$ denotes cross product, $\cdot$ denotes dot product, and $\boldsymbol{v}_x$ and $\boldsymbol{v}_z$ are unit vectors. The matrix $\boldsymbol{R}_i$ is the rotation matrix of the frame attached to link i. This matrix is computed using classic forward kinematics, therefore its value depends on the value of $\boldsymbol{q}$. This equation computes the torque applied by gravity to the axis of rotation $\boldsymbol{v}_z$. The result is separated by monomials, each added as a feature in h. We only use unitary vectors as we assume that information like the position of the center of gravity will be captured by the posterior of $\boldsymbol{\beta}$, i.e. the learned parameters.

These priors are extremely simple as they do not consider friction or coriolis/centrifugal forces. However, they provide important information about the mechanics of the system.

3.4. Experiments

In order to evaluate the performance of the presented model, we performed experiments on a set of simulated systems: Pendulum, Cartpole, Acrobot, and IIWA. We also performed qualitative tests on a toy-dataset to visualize the performance of the EVGP model.

We used Drake [83] to simulate the Pendulum, Cartpole, Acrobot and IIWA systems and obtain the control/sensor data used to train and evaluate the EVGP models. We used the squared exponential function for the covariance kernels. The reason for this choice is that all the experiments involve

continuous systems. The EVGP model was implemented using Tensorflow and the minimization of the negative ELBO loss was done using the ADAM optimizer [55]. The experiments were run in a computer with a single GPU (Nvidia Quadro P5000) with an Intel(R) Xeon(R) CPU (E3-1505M at 3.00GHz).

3.4.1. Experiments on Toy Dataset

The toy dataset is intended to serve as an illustration of the behavior of the EVGP model and visualize the qualitative differences between several GP models. We use a modified version of the toy dataset used in [84] [82]. The dataset[2] is modified as follows:

$$(y, x) \leftarrow (6y + 3x, x)$$

The modification is intended to provide a global linear trend to the data (See Figure 3.5). We use this modification to illustrate how the EVGP is able to model both global and local trends. Global trends describe the overall shape of the function over the entire domain, while local trends describe deviations from the global behavior in specific regions of the domain. Figure 3.5 shows the distribution learned using different versions of a Gaussian Process. Figures 3.5a and Figure 3.5c show the exact posterior distributions for a GP and EGP [80] model, respectively. Figures 3.5b and 3.5d show the variational approximations obtained with a VGP [85] and EVGP model. The standard deviation (black line) is used to visualize the uncertainty. The figures show how the uncertainty grows as we move away from the training dataset.

The original dataset is composed of 200 samples, Figure 3.5 shows that the variational approximations are able to successfully approximate their exact counterparts with as few inducing points as m=10. The position of the inducing points are shown with green crosses.

In this case, the prior-knowledge that we provide to the EVGP is a simple linear function $h(x, \boldsymbol{\beta}) = x\beta_1 + \beta_2$. Figure 3.5d shows how we can use the prior in order to control the global shape of the function. The figure shows how the EGP and EVGP models use the prior knowledge to fit the global behavior of the data (linear) while using the kernels to model the local non-linear behavior.

3.4.2. Learning system dynamics

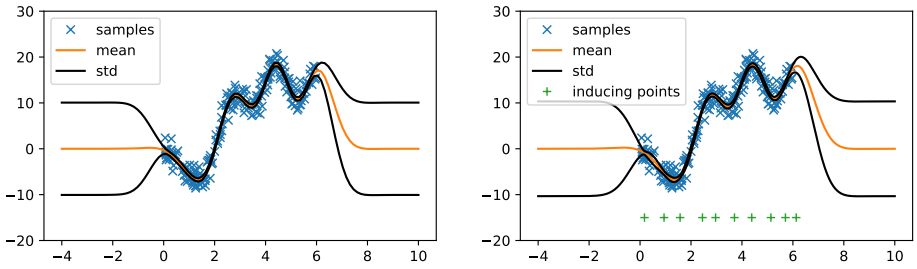
We evaluated the performance of the EVGP model in learning system dynamics using data obtained from simulations of the Pendulum, Cartpole, Acrobot and IIWA systems. Concretely, we evaluated the accuracy of the EVGP model with IF and IFG priors in predicting the next state of the system given the current control inputs and state.

We show the advantages of EVGP regarding generalization and interpretability as follows:

- For generalization, we evaluate all metrics on a testing dataset that has not been used during training. We show the EVGP achieves lower errors on testing data when compared to other models while using smaller training datasets. This means that the EVGP generalizes better to previously unseen data while requiring less data for training.
- For interpretability, we illustrate how to interpret the trained model by visualizing the parameter $\boldsymbol{b}$ from the posterior variational distribution.

Data: to evaluate the difference in generalization, we sampled two different datasets for each system: one for training and one for testing. The experimental procedure is described in Algorithm 2. We used the Pendulum, Cartpole, Acrobot, and IIWA provided in the examples of the Drake library [83]. We did not include process noise or observation noise in the simulated systems. For the Pendulum, Cartpole, and Acrobot, the datasets were sampled by simulating the system using random initial states $\boldsymbol{z}_{[0]} \sim$

[2] Obrained from http://www.gatsby.ucl.ac.uk/~snelson/SPGP_dist.tgz



(a) GP regression (b) VGP regression (m=10)

(c) EGP regression (d) EVGP regression (m=10)

Figure 3.5.: Regression on Toy Dataset. The presented EVGP model provides a tool for the user to control the global shape of the learned function without constraining the complexity. The GP kernels model local non-linear behavior that the global function $h(\boldsymbol{x})$ is unable to model. $h(\boldsymbol{x})$ is linear in this example. We show the contrast of the VGP (b) approximating the GP (a), while the EVGP (d) approximates the EGP (c).

$\alpha\,\mathcal{U}\,(-1,1)$ and random control inputs $\boldsymbol{u}_{[t]} \sim \eta\,\mathcal{N}\,(0,1)$ drawn from uniform and normal distributions, respectively. Several simulations where performed to obtain the trajectories of states and control inputs which where sampled with a period of $\Delta t = 0.03s$. Table 3.1 shows the values of the scales (α, η) that were used to sample the trajectories. In Table 3.1, N refers to the number of sampled trajectories, $|\mathcal{D}|$ refers to the total number of samples. These values were chosen in order to cover at least the range $(-\pi, \pi)$ for the angles on the systems.

For the IIWA, the datasets were obtained by simulating the robot executing a random reference trajectory. The IIWA was controlled by the inverse dynamics controller provided by the Drake library. The controller allowed us to maintain the state of the robot inside a reasonable operating region. The datasets were obtained by simulating the robot starting from a random position $\boldsymbol{q}_{[0]} \sim \alpha\mathcal{U}\,(-1,1)$. A reference trajectory $\boldsymbol{q}_{[t]}$ was generated by adding a random increment from the previous position to the next position in the trajectory $\boldsymbol{q}_{[t+1]} = \boldsymbol{q}_{[t]} + \eta\mathcal{U}\,(-1,1)$. The reference trajectory is used as a reference for the controller. The IIWA is simulated using several random reference trajectories while data of inputs and states is sampled with a period of $\Delta t = 0.006s$.

Baseline: we compare the EVGP model with a standard VGP model [85], a residual VGP (RES-VGP) [36] [85], and a residual Deep Bayesian Neural Network (RES-DBNN) [86] [87]. The VGP model is based on [85] and uses a zero mean prior. The residual VGP and DBNN models assume the system can be approximated as $\boldsymbol{z}_{[t+1]} = \boldsymbol{z}_{[t]} + g_r(\boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]})$, where g_r represents either the VGP or the DBNN model. Approximating residuals is a common approach used to simplify the work for GP and DBNN models [36] [56] [87]. The RES-DBNN model is trained using the local reparameterization trick presented in [86] with Normal variational distributions for the weights. For these set of experiments, we did not consider the exact GP and EGP models given the large number of samples in the training datasets.

Table 3.2 shows the number of inducing points (m) used for the VGP and EVGP models. The table

Algorithm 2 Experimental Procedure

Input: SYSTEM to sample data from. MODEL to be tested.

Step 1: sample TRAIN dataset. Sample N trajectories from SYSTEM using parameters in Table 3.1

Step 2: sample TEST dataset. Sample N trajectories from SYSTEM using parameters in Table 3.1

Step 3: Instantiate MODEL using parameters in Table 3.2

Step 4: train the MODEL using TRAIN dataset and ADAM optimizer

$$\text{MODEL} \leftarrow \text{ADAM(MODEL, TRAIN)}$$

Step 4: test the model using the TEST dataset

$$\text{Error, STD, CR} \leftarrow \text{MODEL(TEST)}$$

Return: Error, STD, CR

Table 3.1.: Parameters used to collect training and testing data

System	α	η	N	Δt (sec)	$\lvert\mathcal{D}\rvert$
Pendulum	[π, 0.5]	1.0	48	0.03	4800
Cartpole	[1.0, π, 0.5, 0.5]	100.0	48	0.03	4800
Acrobot	[π, 1.0, 0.5, 0.5]	0.5	93	0.03	9300
IIWA	0.8	0.03	1500	0.006	225000

Table 3.2.: Number of inducing points and hidden units

	Pendulum	Cartpole	Acrobot	IIWA
VGP	40	100	250	350
RES-VGP	40	100	250	350
RES-DBNN	[15, 15]	[50, 50]	[60, 60]	[250, 250]
EVGP-IF	10	60	150	350
EVGP-IFG	10	60	150	350

also shows the number of hidden units for the DBNN model, where [15, 15] means a two-layer network with 15 units in each layer. We used the LeakyRelu activation function.

Table 3.3 shows the space and time complexity of the models considered in our analysis. Space complexity shows how the memory requirements grow asymptotically as we increase the size of the models. Time complexity shows how the execution time grows asymptotically as we increase the size of the models. In this table, m is the number of inducing points, o is the number of outputs, L is the number of hidden layers, and n is the number of hidden units in each layer. To simplify the analysis, we assume all hidden layers of the DBNN have the same number of hidden units. The table shows that the time complexity of the VGP and EVGP models are governed by the matrix inversion $\boldsymbol{K}_{mm}^{-1}$, which is $O(m^3)$. Because we assume completely independent VGP and EVGP models for each output of the system, their complexity also depends on the number of outputs o. The time complexity of the DBNN model is governed by the matrix-vector product between the weight matrices and the hidden activation vectors, which is $O(n^2)$. For space complexity, the VGP and EVGP require to store m number of inducing points for each output o as we assume independent models for each output. The DBNN

Table 3.3.: Complexity comparison

	Space	Time
VGP	$O(om)$	$O(om^3)$
DBNN	$O(Ln^2)$	$O(Ln^2)$
EVGP	$O(om)$	$O(om^3)$

requires to store the weight matrices (n^2) for each hidden layer L. All models have constant space complexity w.r.t. the training dataset size $|\mathcal{D}|$. Furthermore, all models have linear time complexity w.r.t. $|\mathcal{D}|$ if we assume that training requires to visit each sample in $\mathcal{D}$ at least once.

Metrics: for comparison, we used three metrics: 1) prediction error (Error), 2) predicted standard-deviation ($\|STD\|$), 3) containing ratios (CR). Prediction error is computed as the difference between the dataset values (y) and the model expected estimated output ($\mathbb{E}\left[\hat{y}^{(i)}\right]$). $\|\text{STD}\|$ is computed as the magnitude of the predicted standard deviation:

$$\text{Error} = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\| y^{(i)} - \mathbb{E}\left[\hat{y}^{(i)}\right]\right\| \tag{3.11}$$

$$\|\text{STD}\| = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\|\text{STD}\left(\hat{y}^{(i)}\right)\right\| \tag{3.12}$$

where $|\mathcal{D}|$ is the number of samples in the respective dataset. The expected output ($\mathbb{E}\left[\hat{y}^{(i)}\right]$) for the EVGP model is equal to $\mu_{\hat{\boldsymbol{g}}|\hat{\boldsymbol{x}}}$ in Eq. (3.4). For the DBNN model, the expectation is estimated using Monte-Carlo.

The containing ratios (CR) are the percentage of values covered by the estimated distribution $\hat{y}$. We consider containing ratios for one, two and three standard deviations (CR-1, CR-2, and CR-3 respectively). We expect the best model to have Error and $\|\text{STD}\|$ close to zero, while best containing ratios will be closer to the (68-95-99.7) rule of standard distributions.

Results: Table 3.4 shows the prediction error and CR scores obtained in the testing dataset. EVGP-IF and EVGP-IFG refers to the use of an IF or IFG prior, respectively. We can observe a considerable improvement on the testing error and CR scores when using EVGP models. EVGP models provided the lowest error and best CR scores. We also see a progressive improvement on the testing error when using more detailed priors. The IFG prior provided lower prediction errors when compared with the IF prior. The EVGP-IFG model provided the estimations with the lowest prediction error, with low $\|STD\|$ and best CR scores. Table 3.4 also shows that the EVGP model achieved the best performance using fewer number of parameters.

Figure 3.6 shows a comparison of the prediction error on the test dataset as we increase the number of training samples. For this experiment, we kept the testing dataset fixed while samples were progressively aggregated into the training dataset. The figure shows the mean, max, and min values obtained for four independent runs. Figure 3.6a shows a comparison that includes all models. As expected, the prediction error is reduced as we increase the size of our training dataset. The figure shows that the EVGP provides the most accurate predictions while requiring fewer training samples.

Figure 3.6a also shows how the performance of VGP and RES-VGP plateaus, struggling to take advantage of larger datasets. Although the RES-DBNN performs poorly with small training datasets, the high capacity of the RES-DBNN model allows it to take advantage or large datasets and improve accuracy, reducing the performance gap w.r.t. the EVGP models as more data is available. Thanks to the lower computational time-cost of the RES-DBNN (see Table 3.3), this model can use a larger set of parameters without incurring in excessive training times.

Table 3.4.: Results on Testing Dataset

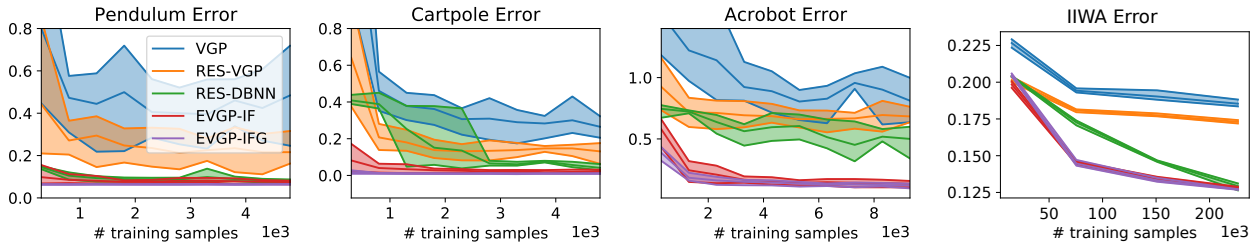
	Model	CR-1	CR-2	CR-3	Error	$\|STD\|$	# Param.
Pendulum	VGP	0.582	0.841	0.910	0.343	0.273	407
	RES-VGP	0.833	0.951	0.974	0.155	0.216	407
	RES-DBNN	0.691	0.908	0.959	0.017	0.010	634
	EVGP-IF	0.936	0.986	0.993	0.013	0.024	119
	EVGP-IFG	0.871	0.965	0.985	**0.005**	0.007	123
Cartpole	VGP	0.505	0.828	0.901	0.188	0.185	2813
	RES-VGP	0.518	0.848	0.924	0.216	0.153	2813
	RES-DBNN	0.336	0.767	0.901	0.044	0.036	6008
	EVGP-IF	0.933	0.980	0.990	0.016	0.184	1733
	EVGP-IFG	0.935	0.976	0.986	**0.011**	0.041	1741
Acrobot	VGP	0.715	0.861	0.913	1.100	2.137	7013
	RES-VGP	0.705	0.846	0.904	0.820	1.428	7013
	RES-DBNN	0.606	0.837	0.899	0.289	0.378	8408
	EVGP-IF	0.794	0.908	0.950	0.151	0.290	4253
	EVGP-IFG	0.712	0.893	0.952	**0.131**	0.251	4269
IIWA	VGP	0.000	0.100	0.574	0.185	0.193	112749
	RESVGP	0.002	0.176	0.672	0.173	0.180	112749
	DBNN	0.000	0.035	0.223	0.129	0.112	143028
	EVGP-IF	0.023	0.396	0.814	0.118	0.138	113337
	EVGP-IFG	0.039	0.448	0.838	**0.117**	0.139	113673

Table 3.5.: Activation Functions and Error comparison on IIWA

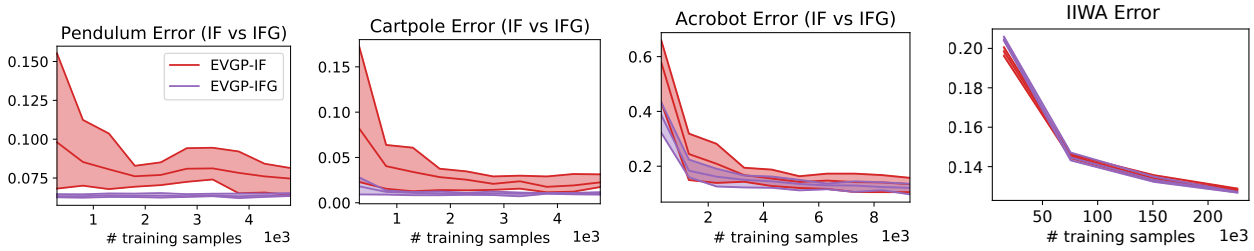
model	Single-step Error	Multi-step Error
RES-DBNN ReLU	0.248	17.423
RES-DBNN SeLU	0.165	7.757
RES-DBNN SiLU	0.180	8.823
RES-DBNN Softplus	0.149	6.542
RES-DBNN LeakyReLU	0.129	5.658
EVGP-IF	0.118	4.812
EVGP-IFG	0.117	4.147

Figure 3.6b shows a scaled version that only considers the EVGP model with different priors. This figure shows that the IFG prior provides more accurate predictions when compared to the IF prior. In the case of the pendulum, the IFG prior provides a highly accurate model of the system, requiring only a small number of training samples. Figure 3.6b also shows how as training data is aggregated, the accuracy gap between IF and IFG priors is reduced. In the case of the IIWA, we observe that the performance of both IF and IFG is very close. The IIWA proved to be the most challenging system. Even when the DBNN provided CR-3 values above 89% for Pendulum, Cartpole, and Acrobot, the DBNN struggled with the IIWA in terms of containing ratios. The advantage of IFG over IF for the IIWA is mostly observed in the containing ratios in Table 3.4.

Figure 3.7 shows a visualization of the EVGP state estimations of the IIWA for multiple time-steps ahead. The multi-step estimations are performed using a standard sequential Monte-Carlo method where particles were used to unroll the EVGP model to predict the state trajectory 45 time-steps ahead [87] [56]. The figure shows that the EVGP is able to provide long-term estimations that follow the dynamics of the physical system. We use the standard deviation to visualize the uncertainty. The figure



(a) Prediction error on testing dataset for different sizes of training datasets.



(b) Detail of prediction error on testing dataset using IF and IFG priors.

Figure 3.6.: Prediction error on testing dataset for increasing number of training samples (shown in thousands, 1e3). The EVGP model achieves lower error on testing data, showing that it is able to generalize better with less training data.

shows how the uncertainty propagates over time.

Table 3.5 shows single step error and multi-step estimation error evaluated on the testing dataset. The multi-step error is an average of the error in Eq. 3.11 computed across the estimated state trajectory for 45 time steps ahead. The single step error is equivalent to Eq. 3.11. The table shows a comparison of the errors between the EVGP and different RES-DBNN models with different activation functions. The table shows that the EVGP provides the lowest single and multi-step error for all models. The table also shows how small single-step errors are accentuated in the multi-step Error. The table also shows that LeakyReLU activation function provides the DBNN model with the lowest error, which is the reason why we choose it for all other experiments.

The priors that we use are extremely simple, they are ignoring friction and coriolis/centrifugal effects. Nonetheless, we observe a considerable performance improvement after providing our data-driven model basic information with the IF and IFG priors. Although we did not include observation or process noise in the training or testing data, we found that including the uncertainty term ϵ_y in the EVGP (eq. 3.5) was essential to obtain a stable training, i.e the error steadily decreasing during training while avoiding NaN results.

Understanding the learned model: one of the advantages of incorporating domain knowledge is that the learned model is easy to interpret by the domain expert. For example, in the case of the Acrobot, the value of the parameter $\boldsymbol{\beta}$ can be visualized to understand and debug what the model has learned. Figure 3.8 shows the value of $\boldsymbol{b}$ learned with the IF (Fig. 3.8a) and IFG priors (Fig. 3.8b).

We observe in Figure 3.8 that the learned parameters follow a similar structure given by the prior (see Eq. 3.10). In our experiments, we did not enforce the sparse structure from the priors, i.e. zero parameters in the prior are allowed to have non-zero values in the posterior.

Figure 3.8a shows that when using the IF prior, the EVGP model compensates for the missing gravity information by assigning negative values to $(\dot{q}_1, q_1)$ and $(\dot{q}_2, q_2)$. The reason for this behavior is that $q_1 \approx \sin q_1$ for small q_1, however this approximation is no longer valid for large q_1. When using IFG priors (Fig. 3.8b), we observe that the model no longer assigns negative values to $(\dot{q}_1, q_1)$ and $(\dot{q}_2, q_2)$. The reason is that IFG provides the values of $sin(q_1)$ and $sin(q_1 + q_2)$ which help to model the effect of

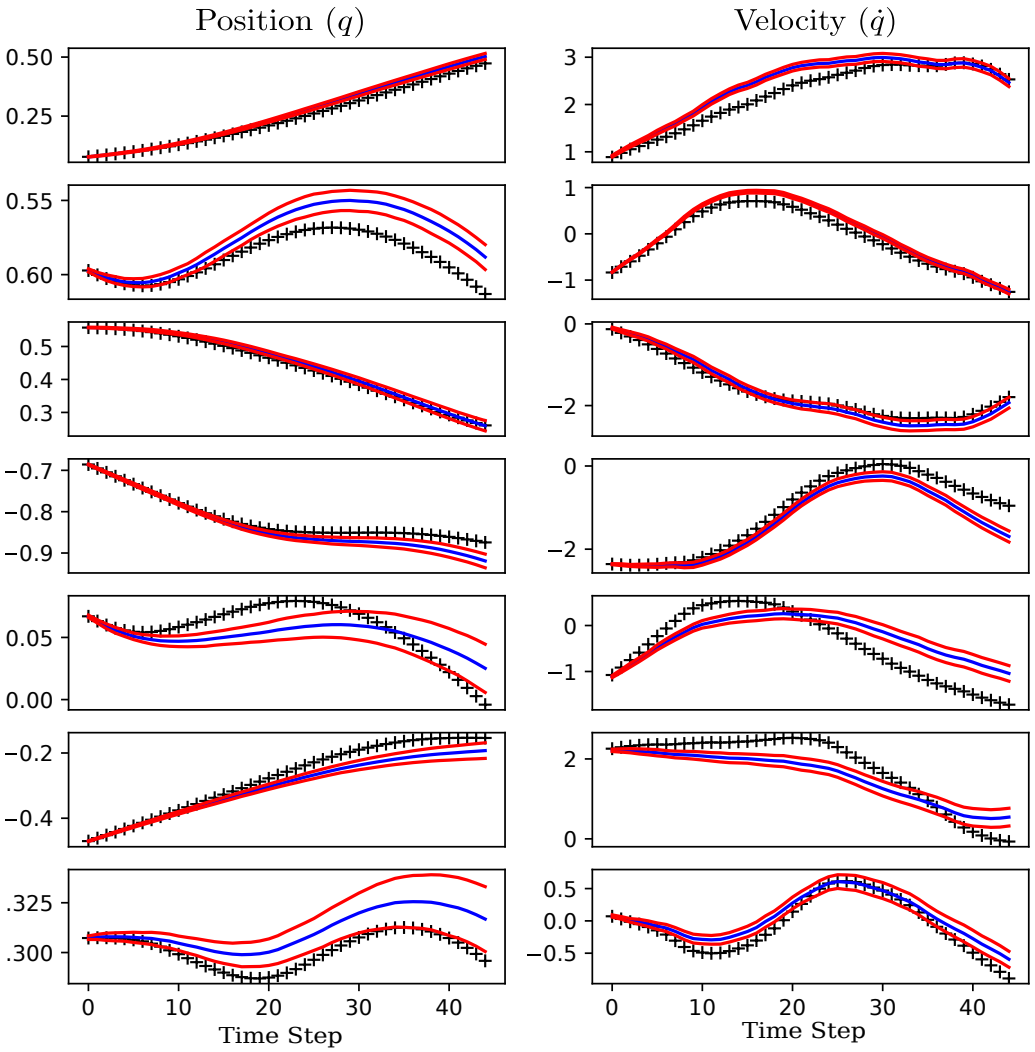
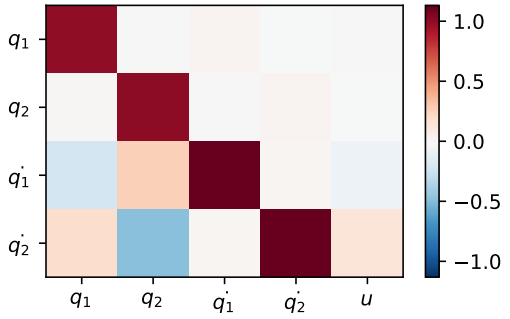


Figure 3.7.: Multi-step (long-term) estimations of the EVGP for the 7-DOF IIWA robot. Black crosses show the real trajectory. Blue shows the mean of the estimated trajectory. Red shows the standard deviation of the predictions to visualize the uncertainty.

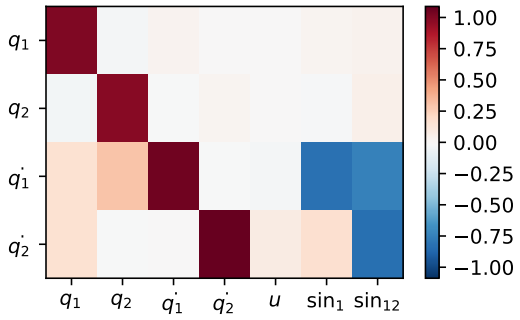
gravity more accurately.

Figure 3.9a shows the visualization of the mean variational posterior $\boldsymbol{b}$ for the IIWA. Similar to the Acrobot, the IIWA posterior has a familiar diagonal structure that resemblances the priors in Eq. 3.10. This observation matches our expectations as we expect the posterior to resemble the prior, which was constructed by placing the pendulum prior on each of the IIWA links. The figure also shows that the lower rows corresponding to $\dot{\boldsymbol{q}}$ have larger values than those corresponding to $\boldsymbol{q}$. This is also expected as most of the inputs should only directly affect the velocity (according to Eq. 3.7). Meanwhile, the position can be estimated by performing discrete integration of the velocities, an observation that is embedded in the priors and evidenced by the diagonal structure and low values of the top rows in Figure 3.9a.

Figure 3.9b shows in detail the visualization of the posterior for $\dot{\boldsymbol{q}} \times \boldsymbol{u}$, scaled by the standard deviation of $\boldsymbol{u}$. The figure shows that elements close to the diagonal have larger values, which makes sense as the input torque has the largest influence on its corresponding link and the adjacent links. Figure 3.9c shows the same posterior but constraining the elements off-diagonal to be zero. Figure 3.9c is an



(a) $\boldsymbol{b}$ using IF prior



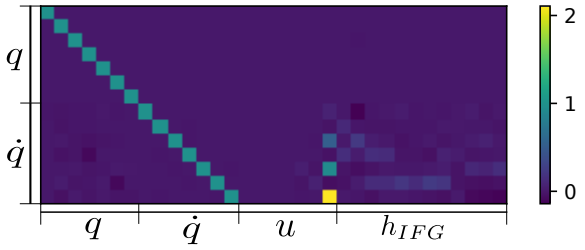
(b) $\boldsymbol{b}$ using IFG prior

Figure 3.8.: Interpreting the EVGP model by visualizing the posterior $\boldsymbol{b}$ (learned from data) for the Acrobot. The value of $\boldsymbol{b}$ has a diagonal structure similar to the prior in Eq. 3.10.

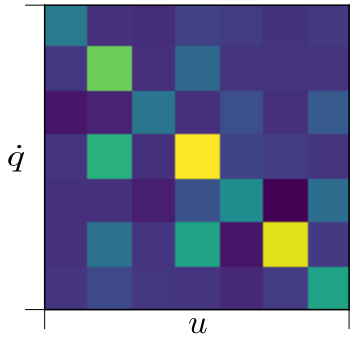
example of how an engineer can manipulate the model by visually inspecting the learned posterior. In our methodology, constraining elements of $\boldsymbol{b}$ is straightforward as $\boldsymbol{b}$ is a trainable parameter whose individual elements can be set to predefined values in each training step. In practice we did not see any improvement of accuracy by constraining zero values in the posterior, but this exemplifies the options available to the engineer thanks to the interpretability of the model provided by relationship between the prior $h(x)^T\beta$ and posterior b.

3.5. Related work

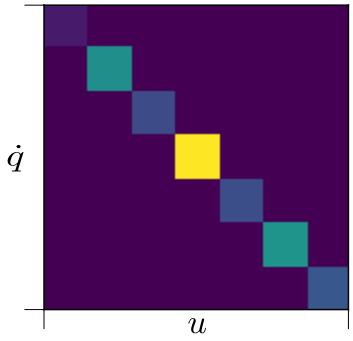
Incorporating prior scientific knowledge in machine-learning algorithms is an ongoing effort that has recently gained increased attention. In [88] LEAP nets are introduced for system identification applied to power systems. The architecture is based on ResNets and uses a set of discrete structural parameters that encode the network topology. Hamiltonian neural networks [89] embeds Hamiltonian mechanics in the architecture of the neural network to ensure that the model follows energy conservation laws. DGNets [90] use a more general formulation for energy-based modeling than Hamiltonian networks and extend the methodology to discrete time. Convolutional neural networks (CNNs) have been used in modeling and simulation applications such as forecasting of sea surface temperature [91] and efficient simulation of the Navier-Stokes equations [92]. Hybrid modeling approaches that combine data and physics based methods are also starting to gain attention in modeling cyber-physical systems [93]. Hybrid models and physics-based loss functions have been introduced in physics-informed neural networks [94] [95] in order to ensure that the trained neural network satisfies energy conservation. In [96] first principles models are combined with machine learning to create deterministic models of process engineering systems. The



(a) Visualization of learned $\boldsymbol{b}$ using IFG prior for the IIWA robot. The visualization shows that the posterior has a diagonal structure similar to the priors in Eq. 3.10.



(b) Detail of $\boldsymbol{b}$ in the region $(\dot{\boldsymbol{q}} \times \boldsymbol{u})$



(c) Detail of $\boldsymbol{b}$ in the region $(\dot{\boldsymbol{q}} \times \boldsymbol{u})$ when constraining off-diagonal elements to zero.

Figure 3.9.: Visualization of the learned value of $\boldsymbol{b}$ for the IIWA 7-DOF Robot.

machine learning model is trained separately to predict the mismatch between a first principles model and the target solution. No training is performed on the first principles model. The solutions of the first principles model and the machine learning model are added to provide the final prediction. In contrast, for the EVGP we model domain knowledge (extracted from simplified first-principles) using a prior probability distribution. This allows us to encode uncertainty in the provided domain-knowledge which in our case is important as we consider domain-knowledge to be imperfect. When we train the EVGP, we use variational inference to obtain a posterior over the parameters given in the prior distribution. The mean of this posterior is represented by $\boldsymbol{b}$. Data is used to simultaneously tune the parameters in the domain-knowlege function $h(\boldsymbol{x})^T\boldsymbol{\beta}$ and train the data-driven GP.

Gaussian Processes (GP) have been used as a general purpose non-parametric model for system identification and control under uncertainty [36] [97]. Previous work has explored using GPs to include partial model information [98]. In [99] the EGP model described in [80] is used in combination with a simplified physics model in a thermal building modelling application. Our work is based on the GP model with explicit features (EGP) presented in [80]. Variations of this model are commonly used in calibration of computer models [100] [101]. To the best of our knowledge, we are the first to apply variational inference to the EGP model in order to embed simplified physics models and improve scalability.

Despite the advantages of GP models for modeling complex non-linear relations with uncertainty, GPs are computationally expensive. A large bulk of research has focused on improving the computational

requirements of GPs. Sparse GP approximation methods are some of the most common approaches for reducing GP computation cost [81]. Bayesian approximation techniques such as Variational Inference provide a rich toolset for dealing with large quantities of data and highly complex models [53] [82]. Variational approximations of a sparse GP have been explored in [82] [85]. In [72] a variational GP model is presented for nonlinear state-space models. In [56] [87], Deep Bayesian Neural Networks (DBNNs) are proposed as an alternative to GPs in order to improve scalability in reinforcement learning problems. Given the popularity of GP models and Variational Inference, there is an increased interest on developing automated variational techniques for these type of models [102] [103].

3.6. Discussion

In this chapter, we wanted to demonstrate how we can use simple concepts from physics that can be scaled and embedded into machine learning models. For this reason, we chose the pendulum as a representative domain problem, demonstrating how it can be used as a building block for 2 DOF systems and expanded to a full 7-DOF manipulator. In a similar manner, the EVGP can be used in other areas such as thermal, electromagnetic, fluids, among others. These systems are usually modeled using differential equations. The EVGP as used in Eq. 3.5 can be used to approximate discretized versions of these systems. The case study that we showed in section 3.3.2 uses an Euler discretization of the differential equation to build the function $h(x)^T\beta$, as shown in Eq.3.7. The presented approach can be extended to other areas by extracting priors from discretized differential equations from the respective area of interest. The extracted priors can then be introduced in the function $h(x)^T\beta$ of the EVGP. If detailed domain knowledge is available, this can be directly incorporated in the function $h(x)^T\beta$. Furthermore, if some parameters values in β are known with high precision (for example, some of them are known to be zero), it is possible to constrain these values in the posterior mean $\boldsymbol{b}$ to ensure that the value is not changed after training.

Another extension of the EVGP that can be explored is the incorporation of non-Gaussian priors and noise. For example, if the engineer knows that certain parameters in the linear prior have to be positive numbers, a distribution such as the exponential distribution may be a better model to use as prior and/or posterior. Evaluating the performance of the EVGP in these types of scenarios and with different priors/posteriors is a possible research direction. Similarly, this chapter assumes normal distribution for the process noise, investigating the effects that non-Gausian noise have in the performance of the EVGP and how to extend the model to use other distributions is another area of improvement.

Although the Gaussian-Process is an universal approximator, the choice of kernel may affect performance in non-continuous systems. In this chapter, all discussed examples were continuous systems. Hence, we used the squared exponential kernel in all experiments as it encodes smoothness assumptions. The presented approach can be extended to non-continuous systems in several ways: 1) discontinuous functions can be added to the domain-knowledge function $h(x)$; 2) a kernel that is adapted to model discontinuities can be introduced instead of the squared exponential [104]; 3) a transformation of the input space can be learned to find a representation that encodes the discontinuity [105].

This chapter presents the EVGP model, a stochastic data-driven model that is enhanced with domain knowledge extracted from simplified physics. The EVGP is a variational approximation of a Gaussian Process which uses domain knowledge to define the mean function of the prior. The priors provided a rough but simple approximation of the mechanics, informing the EVGP of the important structure of the real system. We compared the performance of the EVGP model against purely data-driven approaches and demonstrated improved accuracy and interpretability after incorporating priors derived from simplified Newtonian mechanics. We showed that as we include more detailed priors, the algorithm performance increases. We demonstrated that in the case of smaller training data sets, the EVGP proved its superiority over the purely data driven approaches. Finally, we illustrate how visualizing the learned

parameters can be useful to gain insights into the learned model and hence improve interpretability.

4. Improving interpretability of ML for cyber systems by using graph representations

This chapter is in support of contribution 3: *The Network Transformer, a self-supervised Neural Network model that improves interpretability of cyber anomaly detection models by leveraging graph-based ML representations.*

- *Interpretable model using graph-based ML representations that incorporate the structure of the computer network.*
- *Self-supervised data-driven discovery of representative communication patterns for anomaly detection*
- *Scalability of the presented approach to large datasets*

Papers that preceded this work:

- D. Marino, C. Wickramasinghe, V. Singh, J. Gentle, C. Rieger, M. Manic, "The Virtualized Cyber-Physical Testbed for Machine Learning Anomaly Detection: A Wind Powered Grid Case Study", in IEEE Access, 2021.
- D. Marino, C. Wickramasinghe, B. Tsouvalas, C. Rieger, M. Manic, "Data-Driven Correlation of Cyber and Physical Anomalies for Holistic SystemHealth Monitoring", in IEEE Access, 2021.
- D. Marino, C. Wickramasinghe, K. Amarasinghe, H. Challa, P. Richardson, A. Jillepalli, B. Johnson, C. Rieger, M. Manic, "Cyber and Physical Anomaly Detection in Smart-Grids", in Proc. of the IEEE Resilience Week (RW) 2019, San Antonio, TX, USA, Nov 4-7, 2019.
- D. Marino, C. Wickramasinghe, C. Rieger, M. Manic, "Data-driven Stochastic Anomaly Detection on smart-Grid communications using Mixture Poisson Distributions" in Proc. 45th Annual Conference of the IEEE Industrial Electronics Society, IECON 2019
- D. Marino, C. Wickramasinghe, M. Manic, "An Adversarial Approach for Explainable AI in Intrusion Detection Systems" in Proc. 44th Annual Conference of the IEEE Industrial Electronics Society, IECON 2018, Washington DC, USA, Oct. 21-23, 2018

Characterizing communications in smart-grid distributed control systems is fundamental for understanding the expected behavior and identify abnormal and cyber attack scenarios. In this chapter, we introduce the Network Transformer, an interpretable model that incorporates the graph structure of the communication network in order to perform anomaly detection. This model introduces a self-supervised way to profile the normal communication patterns and identify anomalous behavior using a hierarchical set of features. The presented approach allows to identify anomalies, the devices affected, and the specific connections causing the anomalies, providing a data-driven hierarchical approach to analyze the behavior of a cyber network.

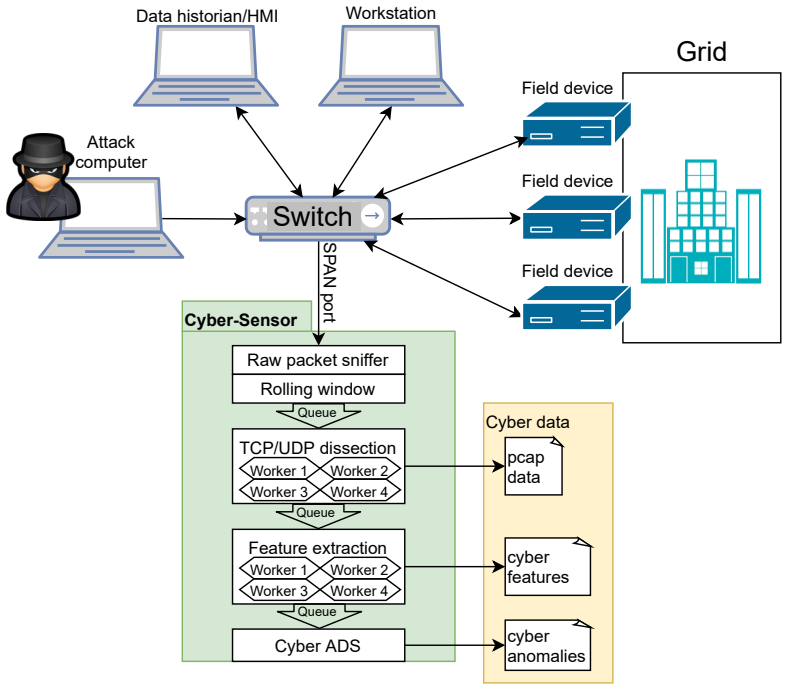


Figure 4.1.: Network traffic monitoring

4.1. Introduction

Cyber-physical systems (CPS) are a collection of interconnected physical and computing resources working together to accomplish a specific task [106]. These systems integrate computations, communications, control, and physical processes into a single system [106]. The operations of these systems are coordinated, controlled, integrated, and monitored by a computing and communication core [107]. CPSs have been increasingly adopted in several industries in order to maximize profit, quality, and resiliency [108]. This integration is particularly notable in the development of the smart grid, with the continuous integration of supervisory control and data acquisition (SCADA) industrial control systems (ICSs) [108].

The smart-grid revolution, fueled by the need for clean, cost-effective electricity generation and distribution, has produced a highly interconnected electrical network [109]. Information and communication technologies (ICTs) play a central role in this revolution as they support communication and control functions in cyber-physical systems. However, the inclusion of ICTs technologies has opened numerous vector attacks which are increasingly targeting critical infrastructure [110]. Machine-Learning-based Anomaly Detection Systems (ADS) provide an important tool to detect such attacks and improve situational awareness [111].

Understanding communication patterns on Cyber-Physical systems is essential to perform anomaly detection and protect infrastructure that relies on ICTs. Machine learning models, such as Artificial Neural Networks, provide an appealing approach for anomaly detection as they can learn communication patterns directly from data. However, these models often follow a black-box design which makes them hard to interpret and extract information [28].

Figure 4.1 shows an example of how data-driven network traffic monitoring is performed in a SCADA network. In this example, a cyber-sensor is connected to a network switch to monitor the communication between devices in the network. The sensor is connected to a switch port analyzer (SPAN port). All incoming and outgoing communication passing through the switch is mirrored to the SPAN port, allowing the cyber-sensor to have access to all packets communicated through the switch. The data acquired through the cyber-sensor is processed first by performing TCP/UDP packet dissection. A

rolling window is used to analyze sections of the data, extracting a series of manually engineered features to be used as inputs to ML anomaly detection systems. The normal behavior of the system is learned by the machine learning algorithms so that any behaviors that are different from previously seen data are flagged as anomalies. During training, the window features characterize the behavior of communications and are used to define the baseline behavior of the system. During deployment, the extracted features are fed into the trained machine learning algorithms (OCSVMs, LOFs, or AEs) for detecting anomalies.

Existing approaches show the capability of ML models to identify abnormal cyber behavior [1, 112, 113, 114]. However, these approaches often lack the interpretability of the results. This is an issue, specially if the approach is applied in monitoring applications. Reporting the detection on an anomaly alone does not provide enough information to an operator to isolate the source of the problem and plan corrective measures. The lack of interpretability of existing approaches is a result of: 1) the black-box models; 2) the features used to train the models. Current approaches for cyber anomaly detection often rely on black-box machine learning models to identify anomalous communication patterns. These models include Autoencoders and LSTM [1, 112, 113, 114]. This is aggravated by the use of features that aggregates information from several devices in the network, resulting in a loss of information from the inherent graph structure of the problem. An example of the aggregated features commonly used in cyber anomaly detection is presented in Appendix D.1. These types of features consist of statistical and temporal features extracted over pre-defined time windows, and they are commonly used in the literature as well as in benchmark datasets such as KDD99 and NSL-KDD [115, 116, 1, 112, 117]. Disaggregating the data features and preserving the graph structure of the problem provides an alternative representation that opens the door for better interpretability and diagnosis.

In this chapter, we present the Network Transformer, a machine learning model that uses graph representations to improve the interpretability of anomaly detection models in network traffic monitoring. The presented model allows not only to identify and anomaly, but it provides a series of hierarchical features that allow to identify devices affected by the anomalies and the connections responsible for the anomalies. The approach leverages self-supervised learning to train the model using unlabeled data. A multi-processing pipeline is presented as a prototype for scalability to large datasets. The rest of the chapter is organized as follows: section 4.2 describes the presented Network Transformer approach; section 4.3 presents the experimental evaluation; section 4.4 presents related work; section 4.5 concludes the chapter.

4.2. Interpretable Anomaly Detection: Network Transformer

In order to create an interpretable model for anomaly detection, we design an approach that leverages the graph structure of computer networks. As shown in Figure 4.2, the idea is to have an abstract model of the problem that is shared by the human and the machine learning model. The abstract model in this case is the graph representation of the computer network being monitored. By embedding this abstract model into the ML approach, we are able to leverage our understanding of the system as a graph in order to extract useful information about the anomalies detected in the system.

Figure 4.3 shows the overview of the presented Network Transformer approach. The approach consists of: a) a packet dissector that groups packages by their respective source and destination addresses; b) an embedded representation obtained using a Transformer Neural Network; c) an Aggregator that extracts a series of hierarchical network features that represent the communication graph. The approach uses the IP address of the devices in the network to represent the nodes in a graph. The packets communicated between the devices are used to represent the edges of the graph. The hierarchical network features are ultimately used for anomaly detection. The hierarchical features include global features representing the entire graph, node features representing each node, and edge features representing individual connections. The approach uses Transformer Neural Networks [118] as they represent the

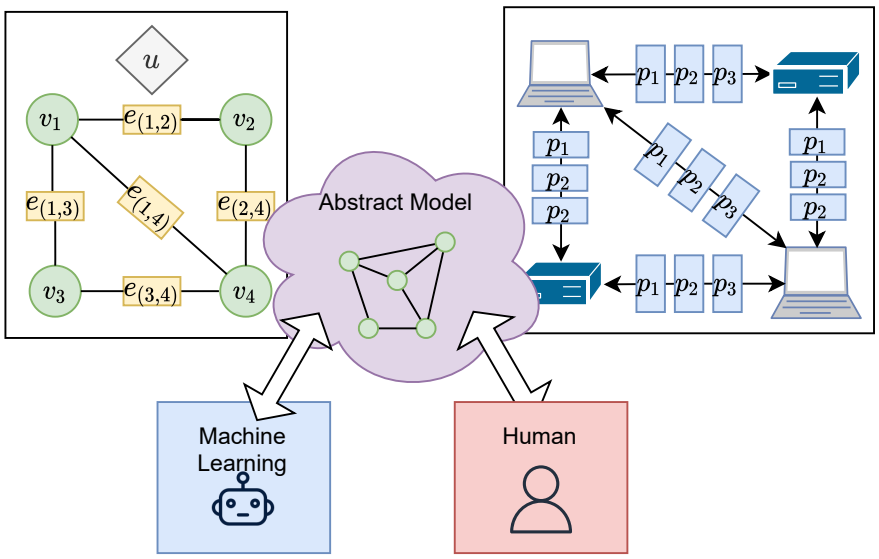


Figure 4.2.: A graph model allows to provide an abstract representation that is shared between human and machine

state of the art for sequence modeling. The hierarchical network features introduced in this chapter are inspired by Graph Networks [33]. The following subsections will expand the description of each one of the aforementioned components.

4.2.1. Graph packet dissection

In order to create a graph network representation, we start by dissecting packet windows in order to identify the source and destination addresses. The address of each device in the network is used to represent a node in the graph. Packets are grouped by their respective source-destination addresses, representing the edges in the graph. Following this approach, each edge in the graph corresponds to a list of packets that are communicated between the respective nodes.

After segmenting the packets by their respective source-destination address, each packet is dissected in order to create an initial feature representation that is suitable for a machine learning model. We considered two sets of features: TCP features and raw Byte features. TCP features are presented in table 4.1. Raw Byte features are presented in table 4.2.

4.2.2. Embedded packet representations, transformer, and self-supervised training

We use a Transformer model [118] in order to encode the list of packets p_k in each edge in to a latent representation $z_{(i,j)}$. The Transformer model is presented in Figure 4.4. The Transformer encodes the lists of packets in each edge of the graph independently, i.e., the Transformer treats each edge independently.

We use a modified version of the Transformer model presented in [118], which is originally used to train language models. Like a sentence composed of a sequence of words, the edges in our network representation are composed of a sequence of packets that are communicated over time. We use the Transformer as it is the current state-of-the-art model for sequence modeling. One of the biggest challenges of sequence models trained by back-propagation has been the problem of the exploding/vanishing gradient [58]. Competing models against Transformers include LSTM (Long-Short Term-Memory) [119] and GRU (Gated Reccurent Units) [120]. Although LSTM and GRU address the problem of exploding/vanishing gradient, these models struggle to capture long-range dependencies [121]. Transformers

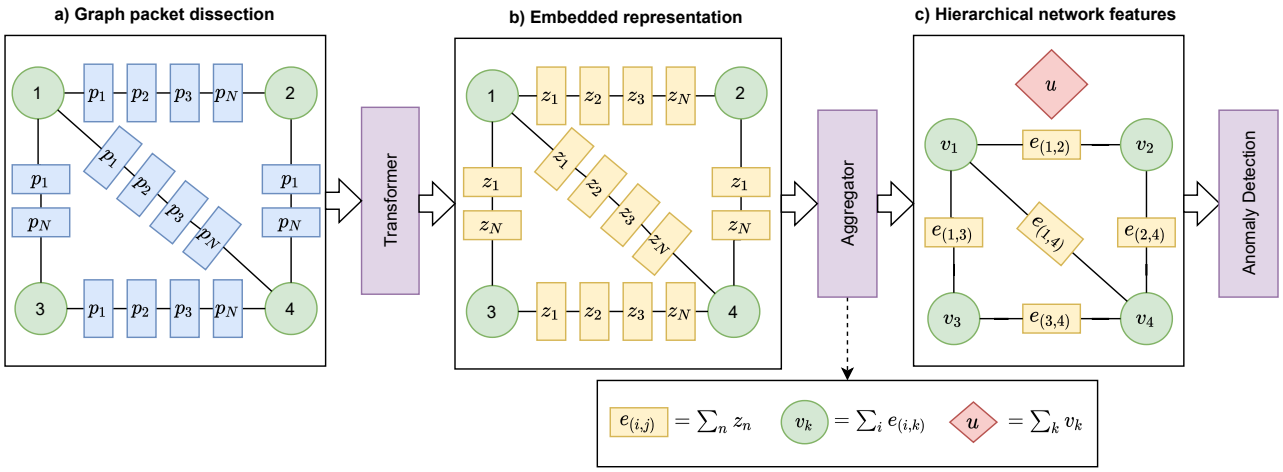


Figure 4.3.: Diagram of the presented Network Transformer (NT) model. The model allows the extraction of a series of hierarchical network features to represent the monitored computer network as a graph.

solve this problem using attention models, which is a type of non-local operation that allows the model to capture long-range dependencies directly, regardless of how relatively distant they are in the input sequence [121]. Using attention operations also improves the efficiency of the training algorithm as sequences can be processed in parallel; this is in contrast to LSTM and GRU models whose inherently sequential nature precludes parallelization [118].

The Transformer model used in this work uses self-supervised learning to obtain the model's parameters. Self-supervised learning refers to a technique where unlabeled data is used to obtain supervisory signals to train the model. In general, it consists of training a model to predict an unobserved part of the data using sections of observed parts of data [122]. The Transformer is trained to predict the next n packets (unobserved) given the sequence of past k packets (observed). As shown in Figure 4.4, the Transformer consists of an Encoder-Decoder network. The Encoder network encodes the input packets into a set of embedded representations that are used by the decoder to predict a series of future packets. This approach allows us to train the Transformer in order to learn an embedded representation of packets that represent the input sequence. This approach also allows us to leverage unlabeled data as the input-output packet sequence can be extracted by dividing unlabeled packet windows in two. Once the Transformer is trained, we use the Encoder network to extract features z_n from the packets that represent the edges of the graph described in section 4.2.1.

As shown in Figure 4.4, the model is composed of four types of layers:

- Linear: applies an affine transformation $f(x) = Wx + b$ of the input using a weight matrix W and a bias vector b.
- Feed Forward: applies an non-linear transformation using stacked layers $f(x) = \sigma(Wx + b)$ where σ is an activation function. We use the ReLU activation function.
- Add & Norm: this layer adds a residual connection [123] followed by Layer Normalization [124]. Residual connections improve gradient propagation while Layer Normalization maintains the mean activation of each sample close to zero, with a standard deviation close to one.
- Multi-Head Attention: this layer consist of several scaled dot-product attention models that evaluate the input in parallel. The scaled dot-product attention consist of a function that performs

Type	Feature	Description
Binary	direction	Direction of the packet: 0 for i to j, 1 for j to i.
Binary	tcp syn	TCP SYN flag
Binary	tcp ack	TCP acknowledgment flag.
Binary	tcp psh	TCP push function.
Binary	tcp urg	TCP urgent flag.
Categorical	layer	Layer of communication: ARP, IP, IPv6, TCP, or UDP.
Categorical	service	Service: DNP3, ftp, http, git, telnet, ssh, x11, etc.
Numerical	len	Size (number of bytes) of the packet.
Numerical	delta time	Difference in seconds between current packet and previous packet.
Numerical	source port	Port used by the source device.
Numerical	dest. port	Port used by the destination device.
Numerical	tcp seq	TCP sequence number.
Numerical	tcp ttl	TCP time to live.
Numerical	tcp window	TCP window.
Numerical	port 22 len	Size of the packet (bytes) when using port 22.
Numerical	port 23 len	Size of the packet (bytes) when using port 23.

Table 4.1.: Packet features when using TCP dissection.

Type	Feature	Description
Binary	direction	Direction of the packet: 0 for i to j, 1 for j to i.
Categorical	layer	Layer of communication: ARP, IP, IPv6, TCP, or UDP.
Numerical	len	Size (number of bytes) of the packet.
Numerical	delta time	Difference in seconds between current packet and previous packet.
Numerical	bytes	List of bytes extracted from the packet (0-512).

Table 4.2.: Packet features when using raw Bytes.

an evaluation of a query matrix (Q) over a series of key (K) value (V) matrices:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_z}}\right)V$$

where d_z is the dimension of the encoded feature representations, which is specified when instantiating the Transformer model. For the Encoder network and the first layer of the Decoder, the values of Q, K, and V are obtained using three Linear layers over the return value of the previous layer. For the second layer of the Decoder, the key-value matrices K and V are obtained using the output of the encoder. This approach follows the same architecture presented in [118]. Both Encoder and Decoder use a mask in order to specify the range of input values that are valid. In the Decoder network, the Masked Multi-Head Attention uses this mask to ensure that the predictions at position n do not depend on future inputs in the sequence.

Figure 4.4 shows $W0$, which is a vector used at the beginning of the target sequence to shift the position of the packets on the right. This is important as the Decoder objective is to predict future packets one at a time. Shifting the input to the right allows the Decoder to predict the next packet given the series of previously predicted symbols, making the Transformer model auto-regressive [118]. In our implementation, the vector $W0$ is a trainable parameter that is learned when training the Transformer model.

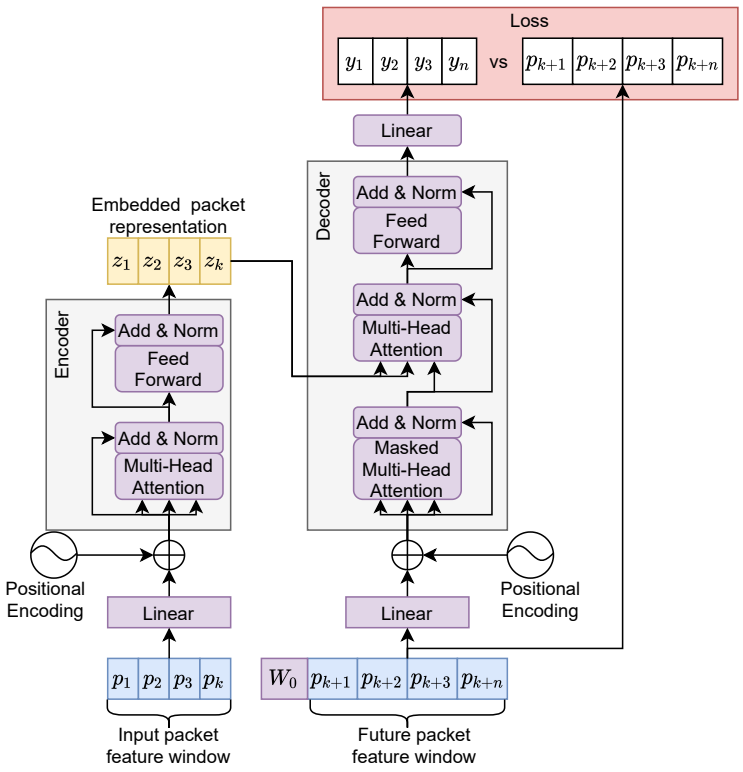


Figure 4.4.: Transformer model and self-supervised training.

The position encoding in Figure 4.4 uses the sine and cosine functions presented in [118]:

$$\text{Positional Encoding}_{(n,2i)} = \sin\left(\frac{n}{10000^{(2i/d_z)}}\right)$$
$$\text{Positional Encoding}_{(n,2i+1)} = \cos\left(\frac{n}{10000^{(2i/d_z)}}\right)$$

where n is the position of the packet in the sequence, i is the dimension position in the packet feature, and d_z is the number of dimensions of the encoded feature representations. Position encoding provides information to the Transformer of the order of the input sequence.

The loss function in Figure 4.4 is used to evaluate the performance of the Transformer network to predict the future window of packets. As shown in Tables 4.1 and 4.2, the packets are represented by a series of Binary, Categorical, and Numerical values. We use this distinction to evaluate the loss depending on the type of the predicted value. Numerical values use a quadratic loss between predicted and target values. Binary and Categorical values use a cross-entropy loss between predicted values and target values.

4.2.3. Hierarchical Graph Features

Figure 4.5 shows the hierarchical graph features used to represent the monitored network. As shown in Figures 4.3, 4.4, and 4.5, graph features are extracted by aggregating the encoded packet representation obtained with the Transformer Encoder. The graph is represented by three types of features: global, node, and edge features. The specification of the features is inspired by Graph Nets [33]. Edge features $e_(i,k)$ represent individual connections between devices. The value of $e_(i,k)$ is obtained by summing the list of encoded packets z_n communicated between node i and k. The encoded packet features z_n are obtained using the Encoder from the Transformer model (see Fig. 4.4). Node features are obtained

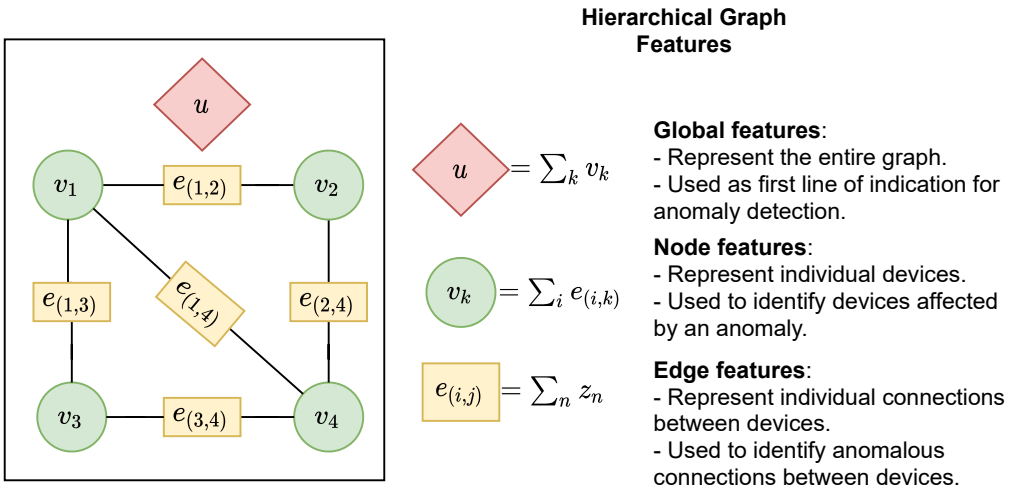


Figure 4.5.: Hierarchical graph features: Global, Node, and Edge features.

by summing the Edge feature values $e_{(i,k)}$ attached to the node. Global features u are obtained by summing all Edge features in the graph.

Graph features are used to represent the computer network for a given window of packets. Graph features provide a way to represent a computer network in different layers of abstraction, using a format that is easy to interpret and exploit in order to extract information. We train anomaly detection algorithms for each level of abstraction (global, nodes, edges) in order to detect anomalies and extract information in different levels of granularity. The algorithms are trained using only data collected during the normal operation of the system. The objective of the anomaly detection algorithms is to flag behavior that deviates from the normal operation. In this chapter, we consider three anomaly detection algorithms: Local Outlier Factor (LOF), One-Class SVM (OCSVM), and Auto-Encoders (AE).

Global features provide the highest level of abstraction by representing the entire device network for the given window of packets. Global features provide a way to characterize the entire behavior of the device network and identify anomalies effectively without dealing directly with individual nodes or connections. This serves as the first indication of anomalous behavior that considers the network as a whole.

Node features are used to characterize the behavior of individual devices. They provide the next level of abstraction after global features. After an anomaly is detected with the global features, node features are used to identify the devices involved in the anomalous event. These devices are detected using an anomaly detection algorithm trained on Node feature data.

Edge features provide the next level of abstraction after Node features. Edge features allow us to identify exactly which connections are exhibiting anomalous behavior. Anomalous connections are also detected using an anomaly detection algorithm trained on Edge features.

4.2.4. Scalability

Network packet data is often characterized by a very high volume of samples. Attacks such as network scans and Denial of Service often flood the computer network with a high volume of packets. As a result, data-driven algorithms need to be designed in order to handle large quantities of data. Performing the graph packet dissection and training of the Network Transformer required the development of a multi-processing pipeline in order to scale to these types of datasets.

Figure 4.6 shows the developed multi-processing pipeline used to train the Network Transformer. The pipeline pre-processes a series of packet capture files (PCAP) into a series of features formatted as tensors (multi-dimensional arrays), ready to be consumed by Tensorflow. The first stage consists

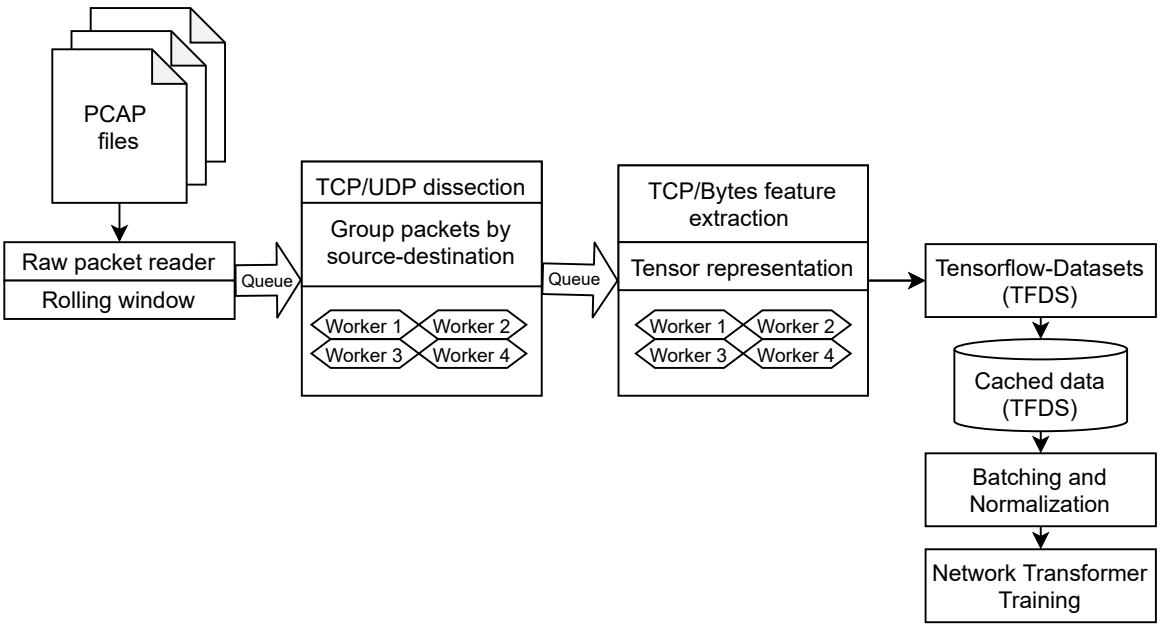


Figure 4.6.: Data pre-processing pipeline

of extracting windows of consecutive raw packets using a rolling window. In our case, we used a window of 30 seconds, but this can be tuned depending on the application. These raw packet windows are introduced in a queue for parallel processing using several workers. The second processing stage consists of TCP/UDP dissection, which groups packets by the source-destination IP address. The third processing stage extracts features according to tables 4.1 and 4.2. These features are grouped using the source-destination address and then packaged in a tensor representation which includes a mask that identifies the set of valid features when having sequences of different lengths. These tensor feature representations are then managed by Tensorflow-Datasets, a library that creates a cache of the pre-processed values in order to be consumed efficiently when training the Network Transformer. The cached data is consumed during training of the Network Transformer, where they are grouped into batches, and numerical features are normalized using a min-max approach. The Network Transformer is trained using Stochastic Gradient Descent, more specifically the ADAM algorithm [55], which scales to very large datasets. Once the Network Transformer is trained, the features can be used as pre-trained features for downstream ML operations without having to re-train the Transformer model. As mentioned before, we use the features for anomaly detection. The pre-processing pipeline in this work was implemented using Python multi-processing library, but the same approach can be horizontally scaled with libraries such as Hadoop or Spark.

4.3. Experiments

This section presents the experimental analysis of the Network Transformer. We used packet captures from a real industrial SCADA system to evaluate the performance of the presented Network Transformer approach. We evaluate the performance on an anomaly detection task and the ability of the model to provide an indication of devices and connections related to an anomaly.

4.3.1. Data Collection

The presented approach was evaluated using a dataset of packet captures collected in an industrial Supervisory Control and Data Acquisition (SCADA) system. Idaho National Laboratory provided the

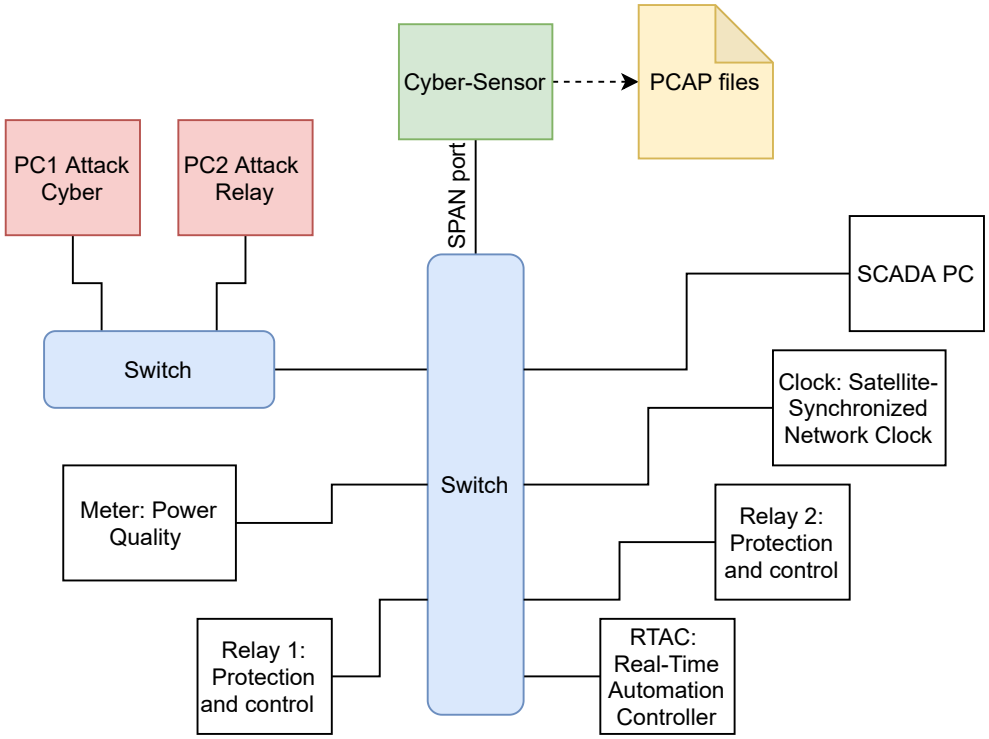


Figure 4.7.: Diagram of SCADA network used to capture the data for experimental evaluation.

dataset and it is based upon work supported by the U.S. Department of Energy's Office of Electricity. The industrial network is presented in Figure 4.7. The network is composed of two attack computers, two protection relays, one power quality meter, one real-time automation controller (RTAC), one satellite synchronized network clock, and one SCADA PC. All devices are connected using two network switches as depicted in Figure 4.7. The RTAC is using the DNP3 protocol to communicate with the relays, the meter, and the SCADA PC. A cyber-sensor is connected to a mirror (SPAN) port in the switch. The SPAN port forwards all the packet data passing through the switch to the cyber-sensor, which stores the data in PCAP files for later analysis. In total, the collected data consisted of 6.036.046 packets.

For experimental evaluation, five types of scenarios were considered:

- Normal scenario: This scenario consists of the normal operation of the system without any cyber disturbance/attack executed.
- Flood attack: Launches a flood of packets using hpin3 command with random source IP address. The attack is launched from the PC1 attack cyber device. The attack targets Relay 1 and Relay 2.
- Scan attack: Launches a network scan using nmap. The attack is launched from the PC1 attack cyber device. The attack targets Relay 1 and Relay 2. Two attacks are launched. The first attack consists of a nmap OS fingerprint scan. The second attack launches a TCP SYN port scan.
- Failed Authentication: An unauthorized user attempts to access remote devices, failing to get access after three attempts. The attack targets Relay 1 and Relay 2 through ssh and telnet. After three unsuccessful login attempts, the relays pulse a series of alarms that are communicated through DNP3.
- Setting change: An unauthorized user successfully accesses Relays 1 and 2 and makes changes in the settings of each device. The setting changes include: changing the current transformer ration

input, phase instantaneous overcurrent level, and relay trip equations. When the user logs into the device, the relay pulses an alarm which is communicated through DNP3 to the RTAC device. After the setting change is executed, the relay pulses another alarm which is also communicated through DNP3.

4.3.2. Global features: anomaly detection

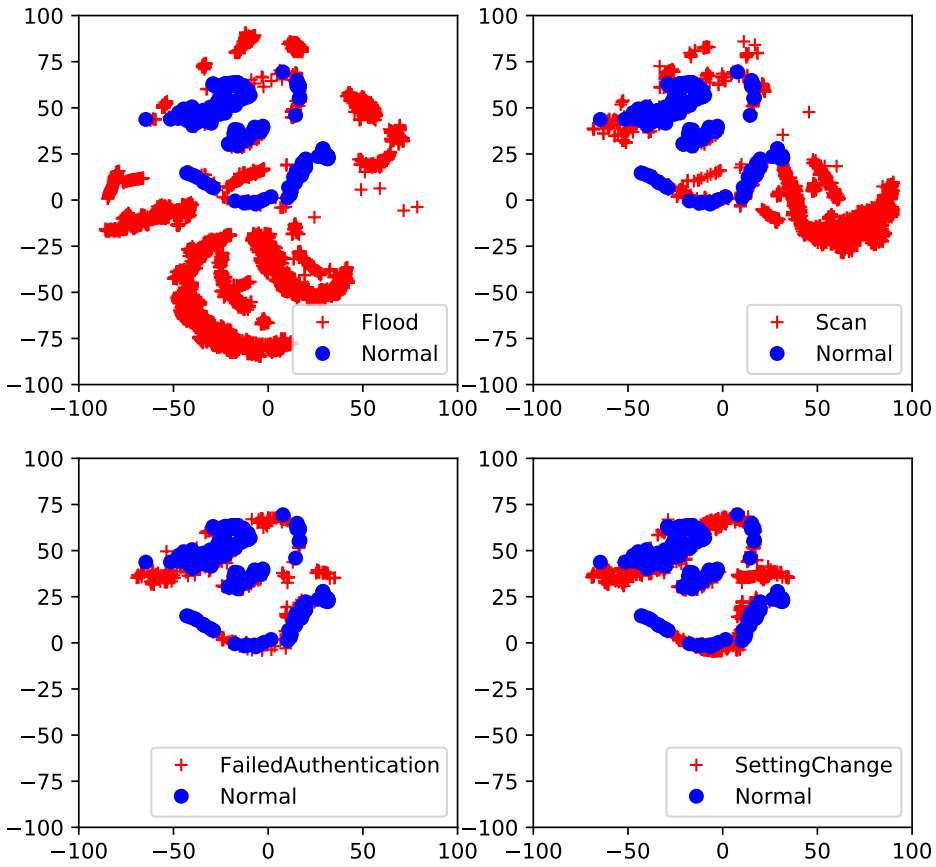


Figure 4.8.: Global features visualized using the T-SNE algorithm

As described in section 4.2.3, global features are used to characterize the behavior of the entire graph. Figure 4.8 shows a visualization of the extracted global features. This visualization uses the T-SNE algorithm [125] to obtain a 2D representation of the high-dimensional embedded features extracted for each scenario of the experiments. The figure shows each attack scenario along with data from normal operations. The figure provides a visual reference of how different each scenario is with respect to the normal behavior of the system. We observe that features from flood and scan scenarios are significantly different from normal behavior. We also observe that the biggest clusters of flood and scan data are located in different areas. Failed authentication and setting change have more subtle differences from normal data. This behavior follows our expectations as Flood and Scan attacks have a large impact on

	FPR train	FPR test	ADR	Flood	Failed Auth	Scan	Setting Change
Baseline LOF	0.0931	0.1009	**0.7486**	**0.8855**	**0.3791**	**0.8439**	**0.7063**
Baseline OCSVM	0.0998	0.1062	0.7023	0.8655	0.3059	0.8105	0.6267
Baseline AE	**0.0092**	**0.0363**	0.6923	0.8648	0.2934	0.7915	0.616148
NT-LOF	0.0896	0.1010	**0.8232**	0.9270	**0.3677**	0.8423	**0.6090**
NT-OCSVM	0.1005	0.1046	0.5124	0.7954	0.2495	0.1131	0.2732
NT-AE	**0.0036**	**0.0336**	0.8249	**0.9609**	0.2848	**0.8492**	0.5128
NTB-LOF	0.0907	0.1019	**0.8471**	0.9514	**0.3465**	**0.8634**	**0.6634**
NTB-OCSVM	0.1001	0.1001	0.2341	0.2805	0.2495	0.1099	0.2804
NTB-AE	**0.0097**	**0.0327**	0.8339	**0.9670**	0.1980	0.8539	0.599073

Table 4.3.: False positive rate and Anomaly Detection Rate for Network-Transformer model.

the network as both introduce a large volume of packets. On the other hand, failed authentication and setting change only introduce a relatively small number of packets on top of normal communication. Moreover, setting change and failed authentication use the same protocol and port (ssh and telnet) as the relay configuration software installed in PC2, which regularly communicates with the relays. This visual representation of the global features serves as an initial understanding of the data, providing a tool to understand the similarity between different scenarios and data.

Table 4.3 shows the anomaly detection results of the Network Transformer (NT). We report results using three anomaly detection algorithms: Local Outlier Factor (LOF), One-Class SVM (OCSVM), and Auto-Encoder (AE). The baseline consist of a series of hand-engineering features used in cyber anomaly detection literature []. The hand-engineering features are described in Appendix D.1. These features consist of a series of TCP features aggregated across packet windows of 30 seconds. NT models refer to the Network Transformer using the TCP packet features from Table 4.1. NTB models refer to the Network Transformer using the raw Bytes packet features from Table 4.2. NT models and NTB models report anomalies detected using packet windows of 30 seconds as well. The table shows the performance measured using False Positive Ratio (FPR) on training and testing datasets and Anomaly Detection Ratio (ADR). Results are measured using 5-fold cross-validation. The FPR measures the rate of anomalies reported for data from the normal scenario. The ADR measures the rate of anomalies reported during attack scenarios. We report ADR measured across all scenarios and for each attack scenario separately. Values of FPR closer to zero and high values of ADR closer to one mean better performance.

Table 4.3 shows that, in general, AE provided results with the lowest FPR with comparable performance on ADR when compared with LOF and OCSVM. NTB and NT provided slightly better FPR with higher ADR when compared with the baseline. We observed a higher ADR for Flood and Scan scenarios using NTB and NT models. However, the baseline provided a slightly better performance on failed authentication and setting change scenarios. For failed authentication, NT-AE and Baseline-AE provided comparable results, with NTB-AE providing the lowest performance in this scenario. For setting change, NTB-AE and Baseline AE provided comparable results, with NT-AE having the lowest performance in this scenario. The baseline and the NT models have specific features indicating activity on ssh and telnet, which helps to explain why they perform better in the failed authentication scenario. On the other hand, setting change is characterized by larger payloads than failed authentication, which helps to explain the better performance of NTB as it includes the raw 512 bytes of the payload.

Table 4.3 shows that the presented NT and NTB models provide comparable or better performance in anomaly detection than the baseline. Furthermore, the baseline has no mechanisms to explore which devices and connections are involved in the anomalies. The following experiments show the capability of the NT model to provide an indication of devices affected and anomalous connections using the Node

and Edge features of the NT approach. Given the leading performance of AE over LOF and OCSVM, the following sections use AE to produce the results for Node features and Edge features.

4.3.3. Node features: Identify devices affected

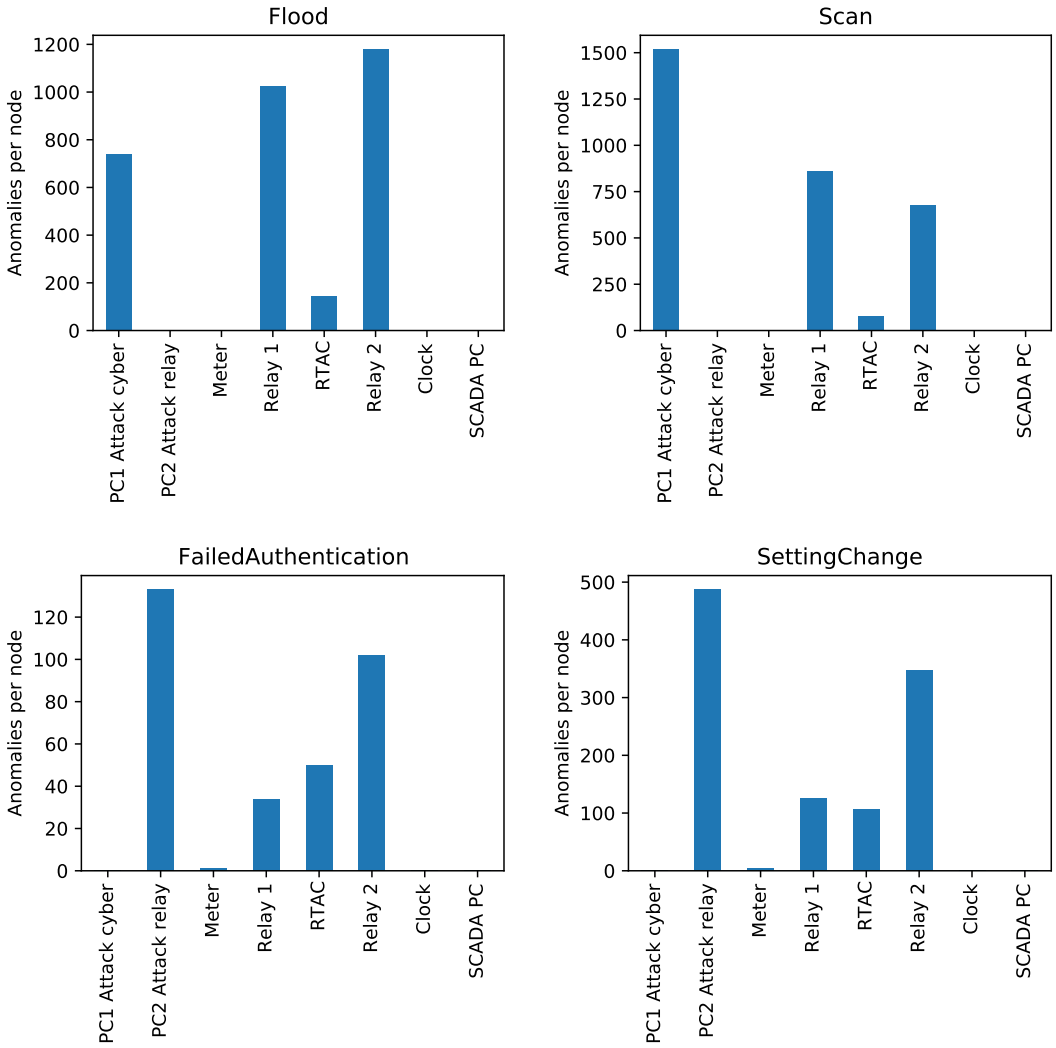


Figure 4.9.: Anomalies in each devices inspected using the NT model. Anomalies are obtained using an AE with Node features extracted using the NT model.

Node features are used to characterize the behavior of each device in the network. These features can be used to identify the devices affected by an anomaly. Figure 4.9 shows the results of devices affected obtained by analyzing the Node features. The figure shows the number of anomalies per device for each one of the attack scenarios.

First, we observe that the presented approach is able to recognize which PC is launching the attack. As described in section 4.3.1, Flood and Scan attacks are launched by PC1 while Failed authentication and Setting Change are launched by PC2. Figure 4.9 clearly shows a high anomaly rate for PC1 in Flood and Scan scenarios while PC2 shows no anomaly. For Failed authentication and Setting Change, PC2 shows a high anomaly rate while PC1 shows no anomalies. Figure 4.9 also shows a high anomaly ratio for Relay1 and Relay2. These relays are the devices targeted by the attacker, evidencing how the

presented approach is able to detect the devices being targeted.

Figure 4.9 also shows a high number of anomalies for the RTAC device during Failed authentication and Setting Change. Although the RTAC device was not targeted by the cyber attacks, the relays communicate with the RTAC whenever there is a failed authentication or a setting change. This communication happens using the DNP3 protocol, and it is described in section 4.3.1. This is an interesting observation because it shows how the presented approach not only detects the attacker and target devices but also detects side effects from the attack.

4.3.4. Edge features: Identify anomalous connections

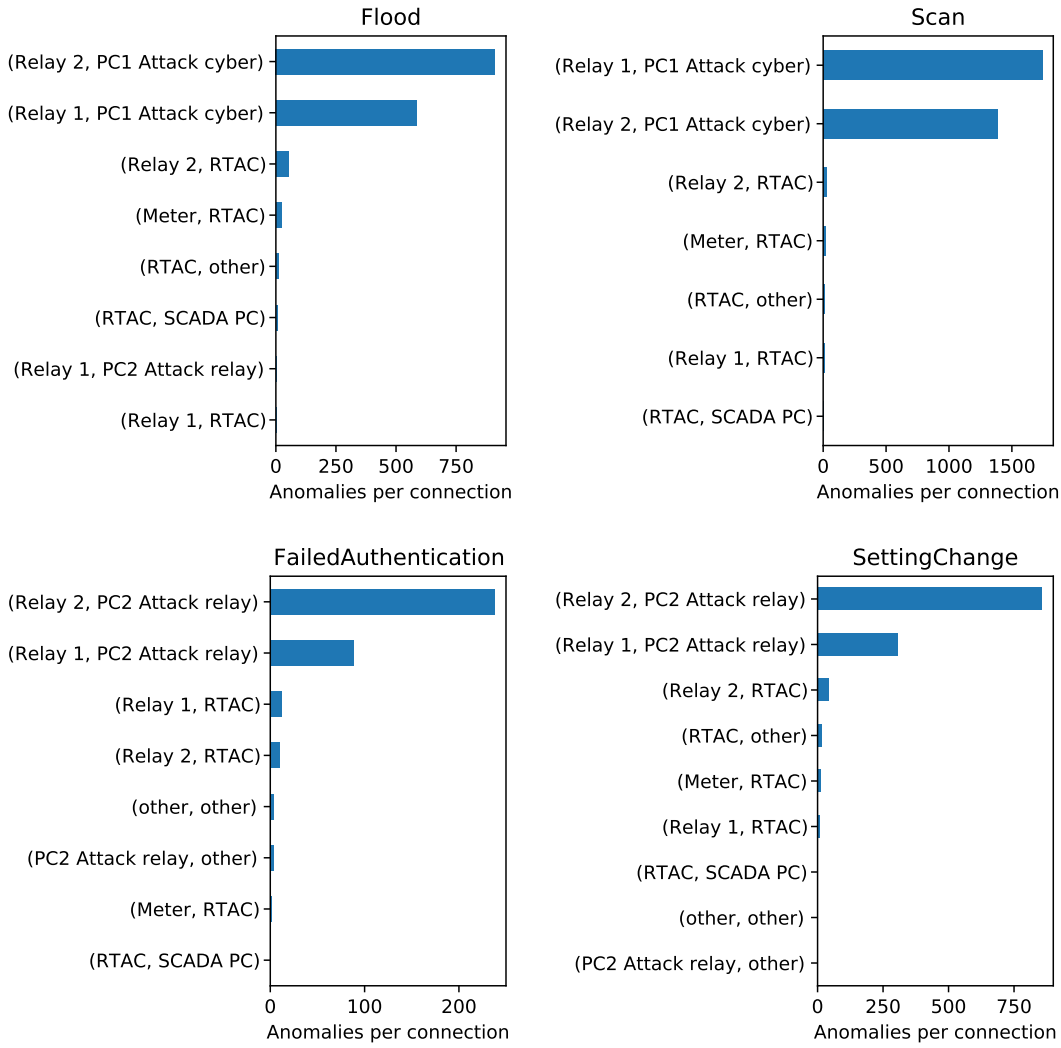


Figure 4.10.: Anomalous connections inspected with the NT model. Anomalies are detected using AE with edge features extracted with the NT model.

Edge features are used to characterize each connection between devices in the network. By running anomaly detection on the Edge features, we are able to identify the connections responsible for an anomaly. Figure 4.10 shows the results of anomalous connections obtained by analyzing the Edge features. The figure shows that the presented approach is able to identify the connections responsible for the anomalies. For the Flood and Scan scenarios, we observe that the approach successfully reports

the anomalous connections between the attacker (PC1) and the relays. For Failed Authentication and Setting Change, the presented approach is able to identify that the anomalous connections involve PC2 and the relays.

Figures 4.8, 4.9, and 4.10 demonstrate how the presented approach allows to extract information from several abstraction layers thanks to the graph structure embedded in the presented approach. Thanks to the equivalence between the learned graph representation and the real device network communication, it is easy for an expert to exploit the learned model in order to extract useful and more detailed information.

4.4. Related work

Cyber-Physical Systems (CPSs) have become core components in modern critical infrastructures [112][15]. The dependency of critical infrastructures on CPSs has made them vulnerable to various attacks, including tampering, invasion, and counterfeiting [15] [126]. Since CPSs consist of multiple components, attacks on a single component of CPS can lead to catastrophic cascading failures[39]. Therefore, it is important to build resilient CPSs with the capability to detect any abnormal behaviors in real-time, increasing the security, credibility, and effectiveness of these systems [126]. A common approach for anomaly detection in computer networks is to use unsupervised machine-learning models with features extracted from log files (usually in text format) which include network traffic, kernel logs, SCADA logs, among others [114, 113].

Anomaly detection is the process of identifying patterns in data that do not represent the expected behavior of a system [127]. Many researchers have successfully used machine learning to detect anomalies in CPSs [127] [15] [128]. In [129], researchers have evaluated various machine learning models to identify power system disturbances. They have presented an evaluation that includes classifiers from various categories such as Probabilistic classification (Naïve Bayes), Rule induction (OneR, NNge, JRipper), Decision tree learning (Random Forests), Non-probabilistic binary classification (SVM), and Boosting (AdaBoost). They have found that even simple models such as OneR and Naïve Bayes have high performance in detecting attacks. Further, they have proposed an ensemble approach that combines JRipper and Adaboost models to detect power system disturbances. In [128], an unsupervised learning algorithm based on Recurrent Neural Networks was successfully used for anomaly detection in a water treatment plant. In [127], an unsupervised learning algorithm has been proposed to learn a signal temporal logic formula (STL) which describes the normal behavior of the system.

In this chapter, we focus on three unsupervised algorithms, namely One-Class Support Vector Machines (OCSVMs), Local Outlier Factor (LOFs), and Autoencoders (AEs). The main reason for using these three algorithms is the difficulty of obtaining precisely labeled data, in the sense that real-world settings emphasize the challenge of dealing with high volumes of unlabeled data. The manual labeling process is time-consuming, expensive, and requires expertise on the data itself [130]. The supervised feature learning is not only unable to take advantage of unlabelled data, but it may also result in biases by relying more on labeled data [131]. Below, a brief description of the anomaly detection algorithms is presented[132]:

- One-class Support Vector Machines: OCSVMs are widely used unsupervised machine learning algorithms for anomaly detection. They are trained using the system's normal behavior, and any unseen behavior is identified as an anomaly or an attack. OCSVMs are extensions of Support Vector Machines (SVMs) [133, 134]. They can learn a decision boundary of a single class. In the case of anomaly detection in CPS, they learn the decision boundary of the normal behavior class. Any behavior which is different from the learned normal behavior will be detected as an outlier [135, 136]. There have been several proposed solutions and enhancements based on OSVMs [137, 138, 139]. Self-adaptive collaborative intrusion detection models have been proposed in

literature which are built using SVMs and decision tree models [140]. These models have shown efficient detection of anomalies in network traffics compared to the use of single SVM models [140].

- Local Outlier Factor: The Local Outlier Factor (LOF) algorithm is a clustering-based unsupervised anomaly detection method that computes the local density deviation of a given data point with respect to its neighbors [141]. It identifies outliers or anomalies as data records that have a significantly lower density compared to its neighbor data points [142]. This has been widely employed for anomaly detection in CPSs[143, 144, 145].

- Autoencoder: Autoencoders (AEs) are widely used neural network architecture which has the capability to learn to encode input data. The architecture of AE consists of two parts: encoder and decoder. The encoder converts input data into an abstract representation which is then reconstructed using the decoder [146, 131]. In anomaly detection, the difference between input and the reconstruction indicates whether the data record is an anomaly or not. AEs have been successfully used in CPSs for malicious code detection, malware detection, and anomaly detection [147, 146].

Self-supervised learning has increasingly gained attention for Out of Distribution (OoD) detection. OoD is an analogous problem to anomaly detection, where the objective is to detect samples that do not belong to the distribution of a given dataset. Self-supervised learning has been increasingly popular on image OoD detection [148, 149, 150]. A popular approach is to train an algorithm to predict the rotation of an image [151]. This approach allows learning deep representations without requiring an expensive labeling process as the rotations can be generated automatically. The trained rotation classifiers have shown remarkable performance on one-class classification, which can be used for anomaly detection [152, 153]. In addition to rotations, other data augmentations such as flipping and translation can be used to automatically generate labeled datasets. These data augmentations can be used with contrastive learning in order to learn deep representations for anomaly detection. In [148], contrastive learning is used to train a model to discriminate if two images, subjected to different data augmentations, belong to the same original instance. The representations learned by contrastive learning are then used as inputs to classic anomaly detection algorithms such as OCSVM.

4.5. Discussion

This chapter presented the Network Transformer, a ML model that uses graph-based representations to incorporate the structure of a monitored computer network. The presented model uses self-supervised learning in order to train the model without the need for labeled data. The Network Transformer provides an approach to extract a series of hierarchical graph features for down-stream ML analysis. In this chapter, we demonstrated the use of the extracted hierarchical features for anomaly detection. The presented approach was evaluated using packet data recorded from a SCADA network. The SCADA network was composed of two attack computers and several industrial devices, including two relays commonly found in real electrical networks. The packet data was collected during normal operation and during the execution of a series of cyberattacks. The Network Transformer was evaluated on an anomaly detection task where the extracted hierarchical features were fed to a series of anomaly detection algorithms. The performance was evaluated by measuring the capability of the presented approach to identify the attacks as anomalies. The Network Transformer performance was evaluated with respect to a baseline model that uses a set of aggregated features. The Network Transformer provided comparable or improved anomaly detection performance while being able to report devices affected and connections compromised, demonstrating its ability for enhanced analytics by providing more detailed information about the anomalies.

Future work can explore the use of the features learned by the Network Transformer for different downstream ML tasks, for example classification of congestion vs DOS attacks. However, is important to keep in mind the practical implications of proposed methodologies. Obtaining labeled data is particularly difficult in cyber-security, specially in safe-critical environments where executing attacks in order to collect data is often not viable. Investigating approaches for self-supervised learning is a promising research direction. One possible approach is to use the difference between transmitted and received packages to learn feature representations.

In this chapter, we also evaluated the performance of the presented Network Transformer using two different sets of packet features: TCP features and raw Byte features. TCP features were obtained by dissecting the packets and extracting the values of TCP flags and other known fields. These features are defined based on prior knowledge of the protocol used. On the other hand, raw Byte features, as the name suggests, use the first 512 bytes as raw features, with minimum use of a-priory knowledge of the packets (minimum dissection). We observed that both TCP and Byte features provided comparable overall results, with TCP providing better results on failed authentication while Byte features provided better results on setting change scenarios. The relative good performance of using Byte features shows a promising research direction on self-supervised feature learning from raw packet data.

The presented approach provided an end-to-end differentiable model for analysing network packet data, starting from potentially raw Byte values up to a global representation of the network graph. The hierarchical design of the approach allows to backtrack and analyze the predictions of the model in different levels of granularity. In the experimental evaluation, we demonstrated the analysis starting from a global representation to analyzing individual nodes and individual connections. The approach can be expanded with existing explainable models such as LIME or SHAP [154, 155], which have been used with Transformers in the past. In this way, because the Transformer Encoder processes potential raw Binary data, the approach can be used as a unified end-to-end methodology for network traffic monitoring that goes from a global view up to individual Bytes.

The presented analysis demonstrates how the graph structure can be exploited to not only identify anomalies but extract useful information of what devices the anomaly is affecting and which connections are responsible for the anomaly. This analysis provides a way to analyze the behavior of a cyber system from different layers of abstraction at different granularity levels, an approach that opens the door for a more interactive and insightful "dialogue" between human analyst and machine learning.

5. Conclusion and Future Work

5.1. Conclusion

This dissertation was motivated by the pressing need for improving the reliability and interpretability of machine learning models. This dissertation met three proposed goals as stated below. The contributions of this dissertation were designed to ensure the scalability of all proposed methodologies to be applicable to the big-data world we live today. We approach reliability by modeling uncertainty using approximate Bayesian methods. Interpretability is approached by embedding domain knowledge in the inductive bias of the presented ML models.

Goal 1: to improve the reliability of Neural Networks in optimal planning tasks by modeling uncertainty

The first goal was fulfilled with the contribution presented in chapter 2, which describes a methodology to improve the reliability of long-term planning using Bayesian Neural Network (BNN) models of system dynamics. We argue and demonstrate that uncertainty modeling is a fundamental tool to improve reliability of neural networks. We use long-term trajectory planning to demonstrate the problems with deterministic neural networks and how modeling uncertainty improves the reliability of the model predictions. The offline planning task serves as a test for reliability as its success depends entirely on the quality of the estimations of the model. As demonstrated in the experiments, the estimations from deterministic DNNs cannot be relied upon when used in offline planning as the predictions do not match the future behavior of the real system. In order to address this reliability problem, we present a methodology that factors the uncertainty of the estimations in the modeling and planning task using BNNs. We presented a method for learning the system dynamics using multi-step predictions. The approach includes measures to ensure stable learning of the dynamics for heteroscedastic models. The experiments demonstrate improved performance in the offline planning task when using the presented multi-step heteroscedastic BNN model. The learned stochastic model was able to accurately estimate the long-term state trajectories together with the uncertainty of the predictions, providing reliable estimations that allowed us to plan completely offline and execute the optimal trajectory in open-loop.

Goal 2: to improve interpretability of machine learning models of physical systems by embedding domain-knowledge

The second goal was fulfilled with the contribution presented in chapter 3, which described the EVGP model, a stochastic data-driven model that improves interpretability by embedding domain-knowledge from physics. The EVGP allows to influence the inductive bias of the model by embedding domain-knowledge in the mean of the prior distribution of a Gaussian Process. In order to ensure scalability to large datasets, we use variational inference to approximate the posterior distribution. We demonstrated how domain-knowledge can be incorporated by embedding simplified physics from Newton's law, applied to rigid-body dynamics (pendulum, cartpole, acrobot, and a robot manipulator). The priors provided a rough but simple approximation of the dynamics, informing the EVGP about important a-priory relationships of the system being modeled. By performing approximate Bayesian inference, we are able to approximate the posterior of the parameters specified by the mean of the prior distribution. Because the mean is specified based on domain-expert knowledge, there is a direct relationship between expert-knowledge and the learned ML model via the posterior distribution of the parameters, providing a link that can be exploited for understanding the learned model. Using this connection between domain-knowledge, prior distribution, and posterior distribution allows the ML model and the user to share

common representations about the problem, improving interpretability of the model. We demonstrated how the posterior of the parameters can be used to interpret the trained EVGP model in the case of the acrobot and the robot manipulator. The experiments also demonstrated improved performance of the EVGP compared to purely data-driven approaches (VGP and BNNs), with improved accuracy and interpretability after incorporating the priors derived from simplified Newtonian mechanics. Furthermore, the experiments showed that increasing the detail of the priors leads to improved performance. The EVGP provided improved accuracy and better containing ratios with smaller datasets when compared to purely data-driven approaches, which serves as indication that embedding domain-knowledge not only improves accuracy but also the quality of the uncertainty estimations.

Goal 3: to improve the interpretability of machine learning models of cyber systems by exploiting the graph structure of computer networks
The third goal was fulfilled with the contribution presented in Chapter 4, which describes the Network Transformer, a self-supervised Neural Network model that improves the interpretability of cyber anomaly detection models by leveraging graph-based ML representations. Compared to other approaches that use manually specified aggregated features, the presented Network Transformer allows us to extract information and analyze the results by leveraging the hierarchical graph structure of learned features. The Network Transformer also leveraged self-supervised learning in order to train the model without requiring the use of labeled data. The pre-processing task of converting raw packet data from PCAP files to tensor representations (multi-dimensional array) was implemented in a multi-processing pipeline. The pipeline provides a prototype to scale the training of the presented model to very large datasets by scaling the computational resources horizontally. The presented approach was evaluated using packet data recorded from a SCADA network which included two attack computers and several industrial control and monitoring devices. The performance was evaluated by measuring the capability of the presented approach to detect a series of cyber attack scenarios launched in the system. In addition, the presented approach also allowed us to identify anomalies with a rate comparable or better to the baseline and provided the means to extract more detailed information about the anomalies. The presented approach allowed us to extract information from the ML model from different abstraction layers without sacrificing accuracy in the anomaly detection task. The graph representation extracted by the Network Transformer enabled us to detect attacks, identify the devices affected, and recognize the specific connections compromised. This analysis was possible thanks to the interpretable structure of the learned hierarchical features, which provide a clear connection with the structure of the real system.

5.2. Current landscape and future opportunities

5.2.1. Reliable ML and uncertainty quantification

High-quality uncertainty quantification holds the promise to greatly impact safe and fair AI [156, 157]. Reliable uncertainty quantification is an important requirement for the development of safe, reliable, and trustworthy AI [158]. Among the topics of active research in recent years is the development of techniques to model (epistemic) uncertainty in Deep Neural Networks. The Bayesian formulation provides a powerful approach to quantify this type of uncertainty, as described in chapters 2 and 3. In this dissertation, we used Monte-Carlo and Variational Inference formulations to perform approximate Bayesian regression with BNNs.

Current theoretical trends in BNNs: From a theoretical perspective, recent work on wide and infinitely-wide BNNs [159, 160, 161, 162] have provided a powerful lens to study BNNs by allowing to perform exact Bayesian regression. Wide and infinitely-wide BNNs have been shown to be equivalent to Gaussian Processes, as such, these neural networks are called NN-GPs in the literature [160]. NN-GPs provide a mechanism to perform exact Bayesian regression by considering weights with a normal

distribution and a variance that scales inversely with the width. In the limit of infinite width, the multivariate central limit theorem allows proving that the prior predictive distribution converges to a Gaussian Process. Under certain conditions, this approach can be used to derive the NN-GP kernel in a closed-form solution, which has a recursive formulation across layers [159, 162]. Although the work in NN-GP has provided valuable theoretical insights understanding the priors and posteriors of BNNs, they have provided limited benefits in practice [162, 160]. Experiments have shown that finite CNNs trained with stochastic gradient-descent (SGD) outperform their NN-GP counterpart, suggesting advantages from SGD training over fully Bayesian parameter estimation [160]. A deeper understanding of the reason for the performance gap between finite NNs and NN-GPs is an interesting research direction that could lead to improvements in using Bayesian approaches in practice.

Uncertainty quantification in practice: From a practical perspective, Deep Ensembles [163] have gained popularity thanks to their simplicity and performance. Deep Ensembles consist of separate NN that are independently trained. Uncertainty estimates are simply obtained by combining their predictive outputs. In the case of classification, this is achieved just by averaging the predictions. For regression, the outputs are combined using a mixture of Gaussians. Empirical results have shown that ensembles provide good uncertainty estimates outperforming BNNs, particularly under dataset shift [163]; however, they incur larger training times, and memory consumption [158]. Our work in chapter 2 is based on Monte Carlo dropout presented in [52]. Dropout may be interpreted as an ensemble model [164], providing a connection with Deep Ensembles. In [165], Ensemble methods were used to improve the performance of reinforcement learning, upholding the motivation and results of the work presented in chapter 2. Recently, deterministic neural networks have been designed to provide uncertainty estimates which outperform Deep Ensembles [158]. In [158], they prove that a deterministic DNN with softmax cannot provide capture epistemic uncertainty, unlike Deep Ensembles. In order to enable the quantification of epistemic uncertainty, [158] fits a Gaussian Mixture Model with one Gaussian per class on the hidden feature space. In order to ensure reliable uncertainty quantification, they ensure sensitivity and smoothness of the bi-Lipschitz constraint using residual connections and spectral normalization [166, 167]. This is a promising research direction as it provides state-of-the-art performance w.r.t. Deep Ensembles without incurring excessive memory and computational costs. Extending this work for regression problems and applying it in reinforcement learning and control in safe-critical systems is an interesting approach for future work.

Testbeds and the ability to compare improvements: Perhaps one of the areas for greater potential impact is in the development of testing and evaluation approaches for BNNs. Recent work introduces Epistemic Neural Networks (ENNs) [157] as a framework to study different BNNs formulations under the same umbrella. The framework provides a promising approach for testing. However, it requires access to high-quality approximations of the posterior, which is unsuitable in many large Deep Learning models applications. [157] also presents a practical testbed for comparing the quality of ENNs. The testbed uses a generative model in order to complement dataset-based benchmark evaluations, which are vulnerable to overfitting. Using a synthetic generative model rather than a dataset with finite samples has the advantage that it can produce an unlimited amount of testing data, avoiding overfitting problems. Using synthetic generative models also allows defining more precise experiments that can be used as benchmarks to measure progress in the field. The application presented in chapter 2 was partially inspired as a way to evaluate the reliability of NN. Instead of using a static training-testing split to evaluate performance, we used iterative trajectory optimization as a way to continuously generate new samples, providing an application where uncertainties estimations heavily influence performance on the task. Future work that focuses on designing benchmark testbed scenarios can have a great impact in the field of uncertainty quantification, given that current approaches have an over-reliance on static datasets.

Metrics for evaluating the quality of uncertainty: The development of metrics to evaluate the performance of the quality of uncertainty quantification is another area of potential improvement. In

this dissertation, we used containing ratios in chapters 2 and 3 in order to provide a metric that was easy to interpret. In [157], their testbed uses KL-divergence between the ENN approximate posterior and an exact NN-GP target posterior computed using neural tangents [168]. In [169] they argue that the KL-divergence is a metric that is not easy to interpret. Even when the KL divergence is apparently small between an exact and an approximate posterior, the approximation can still exhibit bad approximations between the mean and variance, with variance mismatch being particularly sensitive to this problem. [169] proposes the preconditioned Fisher (pF) divergence as an alternative that bounds the 2-Wasserstein distance, providing tight bounds on the mean and variance error. Proposing metrics that are easy to interpret and scalable to compute is fundamental in order to keep track of the progress in the field.

Communicating uncertainty to users and decision-makers: Communicating uncertainty is a fundamental topic of discussion when addressing interpretability of uncertainty-aware ML models. In general, uncertainty is very difficult to calibrate and communicate, which disincentivizes their usage regardless of the potential benefits for decision making. A recent example that highlights how difficult it is to communicate uncertainty has been the lack of effective communication about the effectiveness of vaccines during the COVID-19 pandemic [170]. According to [171], expert knowledge only has practical value if users know how sound it is. To this end, well-calibrated uncertainty communication is fundamental in order to foment trust. However, there is a reluctance by experts to communicate uncertainty [171]. Poorly calibrated uncertainty can lead to over-stated or evasive analysis. One of the reasons for over-stated estimations is the fear that providing uncertainty may give the impression of imprecision or negligence. Experts also fear being misunderstood or being punished for candor. Experts also incorrectly assume non-expert audiences to be incapable of understanding uncertainties and struggle to find means to express uncertainty effectively. Experts can improperly use uncertainty to evade responsibility, leading to evasive analysis. Proper communication of uncertainty requires a tight interaction between experts and decision-makers in order to align goals, incentives, expectations, and the mechanisms to properly interpret the results [171]

The two main contributors of error during decision-making are bias and noise [172]. In this dissertation, we address bias by improving the control over the inductive bias of ML models through the EVGP and by fomenting interpretation of the model using graph representations. Uncertainty estimations provided a way to handle both noise and bias by accounting for areas where data is not available in order to make a well-informed decision (sample bias). Designing ML models that have a closer interaction with humans in the design phase is a potential research direction that has yet to be fully explored. Bias and noise can be reduced if the participants are educated on bias and allowed to participate in mitigation measures [173, 174].

Identifying the source of uncertainty is another important topic in order to communicate effectively with decision-making actors. The framework presented in [158] is careful to disentangle the two types of uncertainty: noise (aleatory) and model (epistemic) uncertainty. In chapters 2 and 3 we were mostly concerned about model uncertainty i.e. uncertainty that derives because of the approximation nature of the learning algorithm and gaps in the training data [87] [36] [56]. We achieve this by using the Variational inference framework. In the case of Chapter 3, although the EVGP is able to handle observation noise when used in regression problems, the recurrent structure used for system dynamics (Eq. 3.5) makes the approach sensible to observation noise. This limitation of recurrent GP formulations has been studied in existing literature [175, 176, 177]. These approaches can be incorporated into the recurrent formulation of the EVGP to explicitly handle observation noise. For example, the direct method from [177] can be incorporated into the EVGP by using moment-matching to estimate denoised states given a sequence of previous noisy measurements. The denoised states can be introduced as inputs to the EVGP and the entire model can be trained using back-propagation. This is an extension that can be carefully explored in future work.

Evaluating trust and reliability in ML: In this dissertation, we see improving trust in ML as the inspiration and ultimate goal for the presented work. However, we did not explicitly measured trust.

Measuring trust is usually a topic covered in fields such as international affairs [178], human factors [179], and social sciences like economics [180], whose methods are outside the scope of this dissertation. A question rises if it is possible to measure trust without a study directly involving humans. An approach to study trust is to identify the main components that are needed in order to positively influence trust. This allows to concentrate efforts in more tangent goals. The OECD follows this approach by proposing a set of Trustworthy AI principles that promote trustworthiness [19]. These principles are listed in the introduction section of this dissertation. Some components of these principles can be studied and measured without directly involving humans. This approach to study trust may present an opportunity to measure improvements in trust without having to directly measure it with human studies.

In this dissertation, we used containing ratios in chapters 2 and 3 as a metric for reliability. Containing ratios measures the ratio of samples that fall outside the prediction of the NNs, providing a metric that gives us an idea of the failure rate of the NN. This dissertation uses the concept of reliability in the broad sense as a motivation for the presented work. The work that we presented is not related to reliability engineering. However, incorporating techniques from reliability engineering can be a promising direction to evaluate performance of NNs.Reliability of the NN can be modeled over time as the system it approximates changes, providing indication on when new data collection and retraining is necessary in order to maintain optimal performance.

5.2.2. Prior-knowledge and inductive bias for interpretable ML

The seminal work on Deep Learning was constructed around applications related to images and sound. These two applications have the particularity of being extremely unstructured, which led to the design of Deep Learning models to have very loose priors as there was no effective way to introduce expert knowledge. As a result, Deep Learning models have been characterized by a lack of interpretability, leading to concerns of bias, accountability, and a lack of trustworthiness [181]. However, the unstructured nature of images and sound is not the norm in many engineering fields, with many applications being heavily structured. Performing a technology transfer between Deep Learning and other engineering fields needs to be aware of this distinction. The technology transfer between traditional Deep Learning to other engineering applications needs to exploit the structure of each application field in order to encourage interpretability of the ML models. In this dissertation, we explored two approaches in chapter 3 and 4 to incorporate the structure of physical and cyber systems (respectively) in the design of the ML model.

Embedding physics knowledge in ML: Recent work on modeling physical systems has focused on leveraging physics knowledge in order to improve the accuracy of ML models when applied to the physics domain. Physics-informed NNs (PINNs) [94] [95] have gain recent attention in the research community. PINNs [94] present a methodology that uses neural networks to solve Partial Differential Equations (PDEs). They introduce the implicit expression of the PDE as part of the loss used to train the model. The loss is a combination of a squared loss and the implicit PDE using data from initial and boundary conditions, as well as data from the PDE solution over the input domain. PINNs have been particularly successful in approximating solutions of high-dimensional PDEs [182] and have been extended with Bayesian NNs to handle forward and inverse PDE problems with noisy data [183].

Although we focused on mechanical systems, the EVGP approach presented in chapter 3 can be extended to most classical physics systems, including electrical, thermal, and hydraulic. All of these systems can be expressed using differential equations, which can be discretized and embedded in the linear EVGP prior using the same approach presented with the mechanical case. The work on PINNs can serve as a reference point to extend the EVGP model to continuous time, eliminating time discretization of the physics model. A deterministic noise-free model can be trained using a physics informed loss directly. In order to include noise and uncertainties, a continuous approach to the long-term estimations presented in chapter 2 can be introduced. This would entail introducing a numerical integrator for-long term estimations. Training can be achieved using the continuous version of Back-Propagation Through time, also known as the adjoint state method. Neural ODEs [184] have revitalized the use of numerical

integrators to train NNs defined as Ordinary Differential Equations (ODE). Neural ODE has also been extended to the Bayesian learning framework [185].

Priors and deep architectures, derived from or derived for: the work on NN-GP has provided valuable insights of the implicit priors of BNNs [159, 160, 161, 162]. The priors have been derived for a number of architectures including CNNs [160], recurrent [186], and attention networks [187]. Recently, a precise characterization of the prior predictive distribution has been presented for the finite-width case with ReLU activations [162]. Most of the work on NN-GPs has been focused on understanding the priors of existing models from a theoretical perspective. However, existing work offers limited flexibility for an user to influence such priors. In chapter 3, we introduced the EVGP model which provides a way to influence the prior distribution by combining domain-knowledge and a Gaussian Process ML model. Instead of relying only on the implicit priors from Deep Learning ML models, exploring alternatives such as the EVGP that allow to explicitly define the priors of ML models can have positive impacts on improving the trust, accountability, and transparency of ML models. We believe that fomenting the interaction between engineer and ML model, especially in the design and training phase, is very important in order to improve understanding, accountability, and trust.

Graph networks as strong inductive bias: The work on graph nets [33], which served as partial inspiration for chapter 4, provides a general-purpose approach to embed relational inductive bias in machine learning models. Graph nets [33] serve as a framework to describe most Graph Neural Networks (GNNs) under the same umbrella. Graph nets have been proven to be extremely successful in introducing strong inductive biases to extract knowledge from learned models, helping to develop more transparent models. The broad application of Graph nets is exemplified in [33] by demonstrating applications in shortest path, sorting, and physical modeling problems. In [188] Graph nets were used to learn models of linked rigid-body systems. In [189] symbolic representations are distilled from a learned deep model. Graph nets provide a powerful tool to introduce expert knowledge, priors, and inductive biases due to the broad application of graphs in real systems and their powerful abstract representations. Exploring their applications in engineering fields is a promising direction in order to improve interpretability while taking advantage of the structure inherent in engineering systems.

5.2.3. Data collection of cyber-physical systems

One of the biggest challenges that we experienced during the development of the Network Transformer in chapter 4 is the lack of available testing and validation methodologies for cyber-physical systems. Engineers have relied on simulation software for decades in order to develop, improve, and understand physical systems. With the increasing introduction of cyber systems in physical infrastructure, it is becoming necessary to introduce models of these technologies in the simulation toolbox available to engineers. Real data is expensive to obtain, especially in cyber security because of privacy and export control concerns. Furthermore, static data does not allow to perform holistic tests of important components such as data acquisition, management, and storage and how they may affect the fidelity of proposed technologies. Cost-effective simulation environments have facilitated recent advances in machine learning and reinforcement learning [190]. This capability is lacking in cyber-security research of cyber-physical systems, with the standard being HIL (Hardware-in-the-Loop) real-time simulations that are prohibitively expensive for many researchers, difficult to maintain, and require extensive training. A cost-effective methodology is necessary in order to evaluate the performance of proposed strategies for machine learning anomaly detection in cyber-physical security. Furthermore, designing a testing methodology that allows integration and tests entire ML pipelines is vital in order to enable a holistic analysis of the entire data-driven pipeline as a whole, including how data is acquired, managed, and analyzed in real-time, with real industrial protocols and ML frameworks. This is fundamental in order to allow for testing ML solutions exactly how they will run in real environments, facilitating the transition to production. In [6], we recently introduced a testbed to simulate malicious cyber-attacks, their effect on the physical system, and detection mechanisms for such disturbances using anomaly detection.

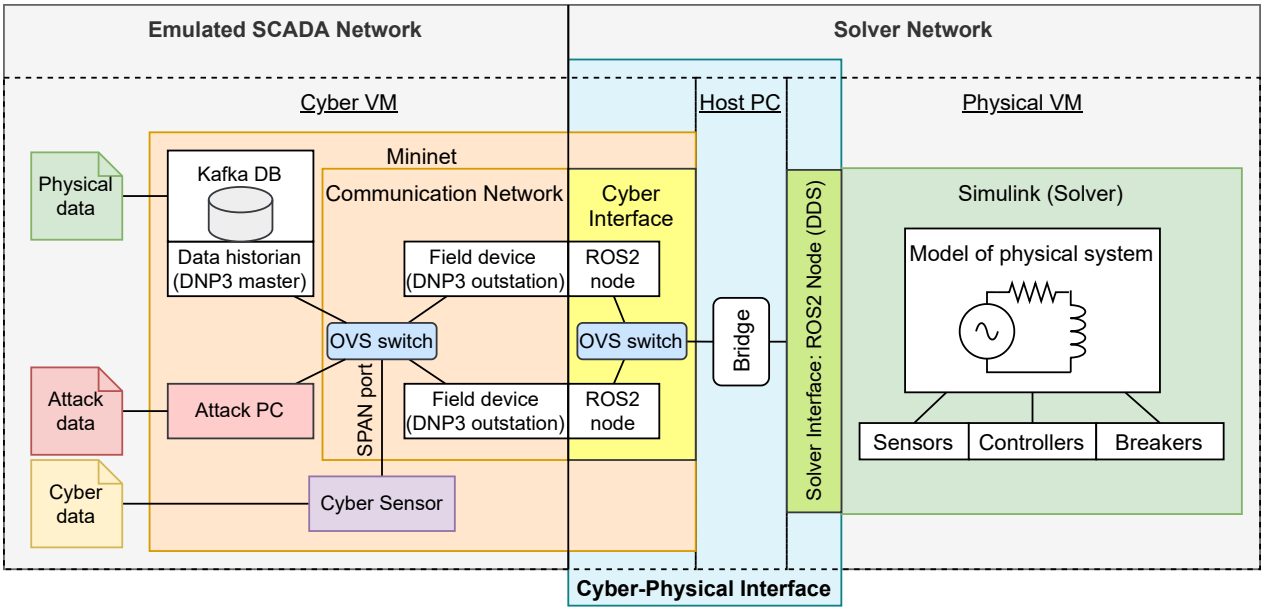


Figure 5.1.: Virtualized Cyber-Physical testbed for data-driven anomaly detection [6]

The testbed architecture is presented in Figure 5.1. Mininet [191] is used to emulate a SCADA network using real industrial protocols, network analysis tools, and industry-leading data-engineering and ML frameworks. MATLAB/Simulink is used to simulate the physical system, using ROS2 as the interface between the network emulation and the physical simulation. The approach provides a cost-effective solution for a holistic analysis of cyber-physical systems by acquiring and analyzing simultaneously cyber and physical data.

The Internet-of-Things (IoT) revolution has energized the development of virtualized environments. Among the technologies in development that could greatly benefit the presented virtualized testbed is incorporating virtualized real-time (deterministic) devices. The Preempt-RT is a patch of the Linux kernel that enables preemption of low priority tasks in order to ensure the reaction to external events (e.g., interrupts) within a predefined time frame. Currently, Preempt-RT is available for every long-term stable version of the Linux kernel since version v2.6.11. Preempt-RT is being supported by the Linux Foundation and it is planned to be merged with the main branch of the Linux kernel [192]. This patch could allow the incorporation of real-time devices in the testbed running on Linux. Project ACRN is developing a hypervisor for IoT embedded applications development. The project is built with real-time and safety-criticality in mind [193]. ACRN allows the deployment of Real-Time Virtual machines, providing a framework for the execution of virtualized real-time devices.

Appendix A. Abbreviations

Acronyms

ADS	Anomaly Detection System.
AI	Artificial Intelligence.
ANN	Artificial Neural Networks.
DBNN	Deep Bayesian Neural Network.
DOF	Degrees of Freedom.
DOE	United States Department of Energy.
DOD	United States Department of Defence.
DOS	Denial of Service attack.
EGP	Explicit Gaussian Process.
ELBO	Evidence lower bound.
EVGP	Explicit Variational Gaussian Process (our proposed method).
GP	Gaussian Process.
IF	inertia-force prior.
IFG	inertia-force-gravity prior.
KL	Kullback-Leibler (divergence).
ML	Machine Learning.
NN	artificial Neural Networks.
RES-DBNN	Residual Deep Bayesian Neural Network.
STD	Standard Deviation.

Symbols

$\oplus$	Concatenation operation.
$\mathrm{Tr}\,()$	Trace operation.
$\mathcal{D}$	Dataset composed of input/output pair of samples $(\boldsymbol{x}, \boldsymbol{y})$.
$\|\cdot\|$	Euclidean norm.

Appendix B. Planning under uncertainty using DBNN

B.1. Variational Distributions

Assuming $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\tau}}$ are composed by L_μ and L_τ hidden layers, respectively, the variational distributions for the parameters are defined as follows:

$$
\begin{aligned}
&w_\mu = \left\{\boldsymbol{A}^{(l)}, \boldsymbol{a}^{(l)}\right\}_{l=1}^{L_\mu}; \quad w_\tau = \left\{\boldsymbol{B}^{(l)}, \boldsymbol{b}^{(l)}\right\}_{l=1}^{L_\tau} \\
&q_\phi\left(\boldsymbol{A}_{[:,i]}^{(l)}\right) = P_{drop}\mathcal{N}\left(0, \sigma_a^2\boldsymbol{I}\right) + (1 - P_{drop})\mathcal{N}\left(\boldsymbol{A}_{\phi[:,i]}^{(l)}, \sigma_A^2\boldsymbol{I}\right) \\
&q_\phi\left(\boldsymbol{a}^{(l)}\right) = \mathcal{N}\left(\boldsymbol{a}_\phi^{(l)}, \sigma_a^2\boldsymbol{I}\right) \\
&q_\phi\left(\boldsymbol{B}_{[:,i]}^{(l)}\right) = \mathcal{N}\left(\boldsymbol{B}_{\phi[:,i]}^{(l)}, \sigma_B^2\boldsymbol{I}\right);\ q_\phi\left(\boldsymbol{b}^{(l)}\right) = \mathcal{N}\left(\boldsymbol{b}_\phi^{(l)}, \sigma_b^2\boldsymbol{I}\right)
\end{aligned}
$$

where $\boldsymbol{A}, \boldsymbol{a}$ represent the weights and bias of $\hat{\boldsymbol{\mu}}$ and $\boldsymbol{B}, \boldsymbol{b}$ the weights and bias of $\hat{\boldsymbol{\tau}}$. The notation $\boldsymbol{A}_{[:,i]}^{(l)}$ represents the ith column of $\boldsymbol{A}^{(l)}$. We follow [52] to define the distribution of $q_\phi\left(\boldsymbol{A}_{[:,i]}^{(l)}\right)$ as a mixture of Gaussians with dropout probability P_{drop}.

The priors for the parameters were defined as follows:

$$
p\left(\boldsymbol{A}_{[:,i]}^{(l)}\right) = \mathcal{N}\left(0, \beta_A^2\boldsymbol{I}\right);\ \ p\left(\boldsymbol{B}_{[:,i]}^{(l)}\right) = \mathcal{N}\left(0, \beta_B^2\boldsymbol{I}\right)
$$

We do not regularize the bias parameters, hence no priors are placed on $\boldsymbol{a}^{(l)}$ or $\boldsymbol{b}^{(l)}$.

To simplify the model, we choose not to optimize over the variance parameters of the variational distributions. Given that the variational distribution and priors are defined using Gaussian distributions, the KL divergence is computed as follows:

$$
KL(q_\phi(w)|p(w)) = \sum_l \frac{\left\|\boldsymbol{A}^{(l)}\right\|_F^2}{\beta_A^2} + \sum_l \frac{\left\|\boldsymbol{B}^{(l)}\right\|_F^2}{\beta_B^2} + K
$$

where K is a constant that is ignored during optimization and $\|\boldsymbol{A}\|_F$ is the Frobenius norm. This expression can be derived directly from the KL divergence between Gaussians and the approximation presented in [52] for dropout.

During inference, we use the mean of the distributions directly as single point estimates. This allows us to represent the following weights and biases directly as variational parameters: $\boldsymbol{a}^{(l)} = \boldsymbol{a}_\phi^{(l)}$, $\boldsymbol{B}^{(l)} = \boldsymbol{B}_\phi^{(l)}$, $\boldsymbol{b}^{(l)} = \boldsymbol{b}_\phi^{(l)}$.

Furthermore, the weight matrices of $\hat{\boldsymbol{\mu}}$ can be re-parameterized using a Bernoulli distribution:

$$
\boldsymbol{A}^{(l)} \sim \boldsymbol{A}_\phi^{(l)}\mathrm{diag}\left(\boldsymbol{\alpha}\right)
$$

where $\alpha_i \sim \mathrm{Bern}\left(P_{drop}\right)$ [52]. Finally, the trainable variational parameters $\phi = \phi_\mu \cup \phi_\tau$ for heteroscedastic models are:

$$
\phi_\mu = \left\{\boldsymbol{A}_\phi^{(l)}, \boldsymbol{a}_\phi^{(l)}\right\}_{l=1}^{L_\mu};\ \ \phi_\tau = \left\{\boldsymbol{B}_\phi^{(l)}, \boldsymbol{b}_\phi^{(l)}\right\}_{l=1}^{L_\tau}
$$

For homoscedastic models, $\phi_\tau = \{\boldsymbol{\tau}_m\}$.

B.2. Loss Gradients

The gradients of $\mathcal{L}_R$ presented in Eq. (2.12) and (2.13) can be obtained using the following differentiation rules [194]:

$$\begin{aligned} \frac{\partial \left(\ln |\boldsymbol{\Omega}|\right)}{\partial \boldsymbol{\Omega}} &= \boldsymbol{\Omega}^{-T}; \;\; \frac{\partial \left(\boldsymbol{v}^T \boldsymbol{\Omega} \boldsymbol{v}\right)}{\partial \boldsymbol{\Omega}} = \boldsymbol{v}\boldsymbol{v}^T \\ \frac{\partial \left(\boldsymbol{v}^T \boldsymbol{\Omega} \boldsymbol{v}\right)}{\partial \boldsymbol{v}} &= \left(\boldsymbol{\Omega} + \boldsymbol{\Omega}^T\right) \boldsymbol{v} \end{aligned} \tag{B.1}$$

Let $\boldsymbol{v} = (\boldsymbol{y} - \hat{\boldsymbol{\mu}})$ and $\boldsymbol{\Omega} = \operatorname{diag}(\hat{\boldsymbol{\tau}})$, i.e. $\boldsymbol{\Omega}$ is a diagonal matrix where the diagonal is equal to $\hat{\boldsymbol{\tau}}$. We can express the loss $\mathcal{L}_R$ from Eq. (2.5) as follows:

$$\mathcal{L}_R = -\log |\boldsymbol{\Omega}| + \boldsymbol{v}^T \boldsymbol{\Omega} \boldsymbol{v}$$

where $\boldsymbol{\Omega}$ is symmetric ($\boldsymbol{\Omega} = \boldsymbol{\Omega}^T$). Using the differentiation rules from Eq. (B.1), the gradients of $\mathcal{L}_R$ can be expressed as follows:

$$\frac{\partial \mathcal{L}_R}{\partial \boldsymbol{\Omega}} = -\boldsymbol{\Omega}^{-1} + \boldsymbol{v}\boldsymbol{v}^T; \;\; \frac{\partial \mathcal{L}_R}{\partial \boldsymbol{v}} = 2\boldsymbol{\Omega}\boldsymbol{v}$$

The gradient $\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\tau}}}$ in Eq. (2.12) is equal to the diagonal of $\frac{\partial \mathcal{L}_R}{\partial \boldsymbol{\Omega}}$. Finally, by using the chain rule we obtain the gradient $\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\mu}}} = -\frac{\partial \mathcal{L}_R}{\partial \boldsymbol{v}}$ which is equal to the gradient shown in Eq. (2.13).

Appendix C. Explicit Variational Gaussian Process

C.1. Abbreviations

EVGP functions and variables

m Number of inducing points in the EVGP. We also use m to indicate that a particular parameter is related to the GP inducing points. For example $\boldsymbol{X}_m$ are the inducing points in the GM.

$\boldsymbol{x} \in \mathbb{R}^I$
Input of the EVGP (vector of size I).

$\boldsymbol{X} \in \mathbb{R}^{DxI}$
Inputs for the EVGP model in Matrix format. Each row of the matrix corresponds to a sample $\boldsymbol{x}^{(i)}$ extracted from the training dataset.

$\hat{\boldsymbol{X}}$ Testing inputs for the EVGP model in Matrix format.

$f(\boldsymbol{x}) \sim \mathcal{GP}\left(0, k(\boldsymbol{x}, \boldsymbol{x}'\right)$
Gaussian Process with zero mean and kernel k.

$\boldsymbol{K}_{\boldsymbol{x}m}, \boldsymbol{K}_{mm}, \boldsymbol{K}_{m\boldsymbol{x}}, \boldsymbol{K}_{\boldsymbol{xx}}$
Matrices that represent the value of the kernel $k(\cdot,\cdot)$ evaluated between: $(\boldsymbol{X}, \boldsymbol{X}_m)$, $(\boldsymbol{X}_m, \boldsymbol{X}_m)$, $(\boldsymbol{X}_m, \boldsymbol{X})$, $(\boldsymbol{X}, \boldsymbol{X})$, respectively.

$g(\boldsymbol{x})$ Denoised output of the EVGP.

y Output of the EVGP.

$\hat{y}$ Output of the EVGP corresponding to the test input $\hat{\boldsymbol{x}}$.

$h(\cdot)$ Explicit function where domain-knowledge is embedded in the EVGP.

$\boldsymbol{H}_{\boldsymbol{x}}, \boldsymbol{H}_m, \boldsymbol{H}_{\hat{\boldsymbol{x}}}$
Matrices where each row represent the value of the function $h\left(\cdot\right)$ applied to samples in $\boldsymbol{X}, \boldsymbol{X}_m, \hat{\boldsymbol{X}}$, respectively.

$\mathcal{L}\left(\phi\right)$ loss function used to train the EVGP model. ϕ are all the trainable parameters in the EVGP.

$\mathcal{L}_{KL}$ KL divergence between the posterior and prior distributions for the parameters $\boldsymbol{f}_m$ and $\boldsymbol{\beta}$.

Parameters

ω Set of model parameters in the EVGP model. These parameters are modeled using probability distributions and include $\boldsymbol{f}_m, \boldsymbol{\beta}$.

$p(\omega)$ Prior distribution of the EVGP parameters.

$p(\omega|\mathcal{D})$ Posterior distribution of EVGP parameters given a training dataset.

$\epsilon_y \sim \mathcal{N}(0, \boldsymbol{\Sigma}_y)$
noise .

$\boldsymbol{f}_m, \boldsymbol{X}_m$ Inducing points in the GP.

$\boldsymbol{\beta}$ Parameters that together with $h(\boldsymbol{x})$ encode the Domain-Knowledge provided to the EVGP.

$p(\boldsymbol{f}_m) = \mathcal{N}\left(0, \boldsymbol{K}_{mm}\right)$
Prior distribution of the parameter $\boldsymbol{f}_m$ defined as a normal distribution with mean $\boldsymbol{\mu}_{\boldsymbol{\beta}}$ and covariance matrix $\boldsymbol{K}_{mm}$.

$p(\boldsymbol{\beta}) = \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\beta}}, \boldsymbol{\Sigma}_{\boldsymbol{\beta}}\right)$
Prior distribution of the parameter $\boldsymbol{\beta}$ defined as a normal distribution with mean $\boldsymbol{\mu}_{\boldsymbol{\beta}}$ and covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\beta}}$.

$p_\phi(\omega)$ variational distribution that approximates the posterior $p(\omega|\mathcal{D})$.

ϕ Set of parameters of the variational approximation which include $(\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{B}, \boldsymbol{X}_m)$.

$p_\phi(\boldsymbol{f}_m) = \mathcal{N}(\boldsymbol{a}, \boldsymbol{A})$
Variational distribution of the parameter $\boldsymbol{f}_m$ defined as a normal distribution with mean $\boldsymbol{a}$ and covariance matrix $\boldsymbol{A}$.

$p_\phi(\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{b}, \boldsymbol{B})$
Variational distribution of the parameter $\boldsymbol{\beta}$ defined as a normal distribution with mean $\boldsymbol{b}$ and covariance matrix $\boldsymbol{B}$.

EVGP for physical system modeling

t Time (discrete).

Δt Sampling interval in seconds.

$\boldsymbol{y}_{[t]} \in \mathbb{R}^o$
Output (vector) of the EVGP at time t. The vector is composed by the output of independent EVGP models.

$\boldsymbol{z}_{[t]}$ State at time t.

$\boldsymbol{u}_{[t]}$ Control input at time t.

$\boldsymbol{x}_{[t]} = \boldsymbol{z}_{[t]} \oplus \boldsymbol{u}_{[t]}$
Input of the EVGP at time t.

J, γ_i Parameters used to define the structure and values of $\boldsymbol{\beta}$. They are used only to illustrate how to embed domain-knowledge in $h(\boldsymbol{x})^T\boldsymbol{\beta}$, but they ultimately become part of $\boldsymbol{\beta}$.

q_i Position of the link i.

R_i Rotation matrix of the frame attacked to a link i.

$\boldsymbol{v}_x, \boldsymbol{v}_y, \boldsymbol{v}_z$
unitary vectors in axes x, y, and z, respectively.

$\boldsymbol{v}_g$ unitary vector in the direction of gravity.

C.2. Variational Inference

In a Bayesian learning approach, we model the parameters ω of our model $p(\boldsymbol{y} \mid \boldsymbol{x}, \omega)$ using probability distributions. The parameters are given a prior distribution $p(\omega)$ that represents our prior knowledge about the model before looking at the data. Given a dataset $\mathcal{D}$, we would like to obtain the posterior distribution of our parameters following the Bayes rule:

$$p(\omega|\mathcal{D}) = \frac{p(\mathcal{D}|\omega)p(\omega)}{p(\mathcal{D})}$$

Variational Inference (VI) provides a tool for approximating the posterior $p(\omega|\mathcal{D})$ using a variational distribution $p_\phi(\omega)$ parameterized by ϕ. In other words, with VI we find the value of ϕ such that $p_\phi(\omega) \approx p(\omega|\mathcal{D})$. The parameters ϕ of the distribution $p_\phi(\omega)$ are found by maximizing the Evidence Lower Bound (ELBO) between the approximate and real distributions:

$$\phi \leftarrow \arg\max_\phi \mathop{\mathbb{E}}_{p_\phi(\omega)} [\log p(\mathcal{D} \mid \omega)] - D_{KL}(p_\phi(\omega) \,||\, p(\omega))$$

Maximizing the ELBO is equivalent to minimizing the KL divergence between the variational distribution and the real distribution.

Having obtained the variational approximation $p_\phi(\omega)$, we can approximate the predictive distribution $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D})$ as follows:

$$p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \mathop{\mathbb{E}}_{p(\omega|\mathcal{D})} [p(\boldsymbol{y}|\boldsymbol{x}, \omega)] \approx \mathop{\mathbb{E}}_{p_\phi(\omega)} [p(\boldsymbol{y}|\boldsymbol{x}, \omega)]$$

which is the approximated variational predictive distribution $p_\phi\left(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{X}}\right)$ (see section 3.2.3):

$$p_\phi\left(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{X}}\right) = \mathop{\mathbb{E}}_{p_\phi(\omega)}\left[p\left(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{X}}, \omega\right)\right]$$

C.3. EVGP ELBO

Given the training dataset $\mathcal{D} = (\boldsymbol{y}, \boldsymbol{X})$, the parameters ϕ of $p_\phi(\omega)$ are learned by minimizing the negative Evidence Lower Bound (ELBO). For the EVGP, the negative ELBO takes the following form:

$$\mathcal{L}_1(\phi) = -\mathop{\mathbb{E}}_{p_\phi(\omega)}\left[\ln \mathop{\mathbb{E}}_{g|\omega}\left[p\left(\boldsymbol{y} \mid \boldsymbol{g}\right)\right]\right] + \mathcal{L}_{KL} \tag{C.1}$$

where $\mathcal{L}_{KL}$ denotes the KL divergence between the variational posterior and the prior $\mathcal{L}_{KL} = D_{KL}\left(p_\phi(\omega) \,||\, p(\omega)\right)$ Note that the inner expectation in Eq. (C.1) is taken w.r.t. $g \mid \omega$, presented in Eq. (3.2). Following a similar approach than [85], we apply Jensen's inequality in the inner expectation of Eq. (C.1):

$$\log \mathop{\mathbb{E}}_{g|\omega}\left[p\left(\boldsymbol{y} \mid \boldsymbol{g}\right)\right] \geq \mathop{\mathbb{E}}_{g|\omega}\left[\log p\left(\boldsymbol{y} \mid \boldsymbol{g}\right)\right]$$

$$\mathop{\mathbb{E}}_{g|\omega}\left[\log P\left(\boldsymbol{y} \mid \boldsymbol{g}\right)\right] = \log \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{\mu}_{\boldsymbol{g}|\omega}, \boldsymbol{\Sigma}_y\right) - \frac{1}{2}\mathrm{Tr}\left(\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{\boldsymbol{g}|\omega}\right)$$

where $\boldsymbol{\mu}_{\boldsymbol{g}|\omega} = \boldsymbol{H}_x\boldsymbol{\beta} + \boldsymbol{\mu}_{\boldsymbol{f}|\omega}$, $\boldsymbol{\Sigma}_{\boldsymbol{g}|\omega} = \boldsymbol{\Sigma}_{\boldsymbol{f}|\omega}$. This allows us to express the ELBO in a way that simplifies the computation of the expectations w.r.t. the parameters ω. Now, the variational loss in Eq. (3.3) can be obtained by simply computing the expectation w.r.t. the model parameters:

$$\begin{aligned}
\mathcal{L}(\phi) &= \mathop{\mathbb{E}}_{p_\phi(\omega)}\left[\mathop{\mathbb{E}}_{g|\omega}\left[-\log p\left(\boldsymbol{y} \mid \boldsymbol{g}\right)\right]\right] + \mathcal{L}_{KL} \\
&= -\mathop{\mathbb{E}}_{p_\phi(\omega)}\left[\log \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{\mu}_{\boldsymbol{g}|\omega}, \boldsymbol{\Sigma}_y\right) - \frac{1}{2}\mathrm{Tr}\left(\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{\boldsymbol{g}|\omega}\right)\right] \\
&\quad + \mathcal{L}_{KL} \\
&= -\log \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{H}_x\boldsymbol{b} + \boldsymbol{K}_{xm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{a}, \boldsymbol{\Sigma}_y\right) \\
&\quad + \frac{1}{2}\left[\mathrm{Tr}\left(\boldsymbol{M}_1\boldsymbol{A}\right) + \mathrm{Tr}\left(\boldsymbol{M}_2\boldsymbol{B}\right) + \mathrm{Tr}\left(\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{\boldsymbol{f}|\omega}\right)\right] \\
&\quad + \mathcal{L}_{KL}
\end{aligned}$$

where $\boldsymbol{M}_1 = \left(\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mx}\right)\boldsymbol{\Sigma}_y^{-1}\left(\boldsymbol{K}_{xm}\boldsymbol{K}_{mm}^{-1}\right)$, and $\boldsymbol{M}_2 = \boldsymbol{H}_x^T\boldsymbol{\Sigma}_y^{-1}\boldsymbol{H}_x$. The value of the KL-divergence is simply the sum of the divergence for both parameters:

$$\begin{aligned}
\mathcal{L}_{KL} =& D_{KL}\left(\mathcal{N}\left(\boldsymbol{a}, \boldsymbol{A}\right) \,||\, \mathcal{N}\left(0, \boldsymbol{K}_{mm}\right)\right) \\
&+ D_{KL}\left(\mathcal{N}\left(\boldsymbol{b}, \boldsymbol{B}\right) \,||\, \mathcal{N}\left(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta\right)\right)
\end{aligned}$$

C.4. Mini-batch optimization

In order to make the model scalable to very large datasets, the ELBO can be optimized using mini-batches. Following [52], assuming the samples are i.i.d., the loss for a mini-batch $(\boldsymbol{y}, \boldsymbol{X})$ composed of $|\boldsymbol{X}|$ number of samples can be expressed as follows:

$$\mathcal{L}(\phi) = -\frac{1}{|\boldsymbol{X}|}\mathop{\mathbb{E}}_{p_\phi(w)}\left[\ln\left(p(\boldsymbol{Y}|\boldsymbol{X}, w)\right)\right] + \frac{1}{|\mathcal{D}|}\mathcal{L}_{KL}$$

where $|\mathcal{D}|$ is the total number of samples in the training dataset.

Appendix D. Cyber Anomaly Detection

D.1. Window-based TCP packet features

Table D.1 shows the features extracted from PCAP files using the windowing technique presented in [1]. These features consist of statistical and temporal features extracted over pre-defined time windows. The features are used as inputs for machine learning algorithms, which in this dissertation include One-Class SVMs, Local-Outlier Factor, and Autoencoders.

Table D.1.: Window based TCP packet features [1]

Feature Name	**Feature Description**
Packet_rate	Number of packets within a window
Num_src_IP	Number of different source IP addresses
Num_dst_IP	Number of different destination IP addresses
Num_src_port	Number of different source ports
Num_dst_port	Number of different destination ports
Min_data_length	The minimum data length of packets
Max_data_length	The maximum data length of packets
Avg_data_length	The average data length of packets
Min_win	The minimum window size of packets
Max_win	The maximum window size of packets
Avg_win	The average window size of packets
Min_time_intv	The minimum time gap between packets
Max_time_intv	The maximum time gap between packets
Avg_time_intv	The average time gap between packets
Min_pkt_src	The minimum number of packets per single source IP
Max_pkt_src	The maximum number of packets per single source IP
Avg_pkt_src	The average number of packets per single source IP
Min_pkt_dst	The minimum number of packets per single destination IP
Max_pkt_dst	The maximum number of packets per single destination IP
Min_ttl	The minimum time to live value of packets
Max_ttl	The maximum time to live value of packets
Avg_ttl	The average time to live value of packets
Num_byt	Number of bytes transmitted by packets
Same_src_dst	Number of packets with src IP = dst IP
Same_ports	Number of packets with src port = dst port
Same_src_src_port	Number of unique of src IP and src port combinations
Same_src_dst_port	Number of unique of src IP and dst port combinations
Same_dst_src_port	Number of unique of dst IP and src port combinations
Same_dst_dst_port	Number of unique of dst IP and dst port combinations
Same_IP_port	Number of packets with src IP = dst IP and src port== dst port
Num_urg	Number of urgent packets
Num_syn	Number of sync packets
Num_arp	Number of arp packets

Appendix E. List of publications by the author

Journals:

1. D. Marino, C. Wickramasinghe, V. Singh, J. Gentle, C. Rieger, M. Manic, "The Virtualized Cyber-Physical Testbed for Machine Learning Anomaly Detection: A Wind Powered Grid Case Study", in IEEE Access, DOI: 10.1109/ACCESS.2021.3127169, 2021

2. D. Marino, C. Wickramasinghe, B. Tsouvalas, C. Rieger, M. Manic, "Data-Driven Correlation of Cyber and Physical Anomalies for Holistic SystemHealth Monitoring", in IEEE Access, DOI: 10.1109/ACCESS.2021.3131274, 2021

3. S. Vazquez, D. Marino, E. Zafra, M. Valdes, J. Rogriguez, L. Franquelo, M. Manic, "An Artificial Intelligence Approach for Real-Time Tuning of Weighting Factors in FCS-MPC for Power Converters", in IEEE Transactions of Industrial Electronics, DOI: 10.1109/TIE.2021.3127046, 2021

4. D. Marino and M. Manic, "Physics Enhanced Data-Driven Models with Variational Gaussian Processes," in IEEE Open Journal of the Industrial Electronics Society, DOI: 10.1109/OJIES.20 21.3064820, 2021

5. C. S. Wickramasinghe, K. Amarasinghe, D. Marino, C. Rieger, and M. Manic, "Explainable Unsupervised Machine Learning for Cyber-Physical Systems," in IEEE Access, vol. 9, pp. 131824-131843, 2021, DOI: 10.1109/ACCESS.2021.3112397.

6. C. S. Wickramasinghe, D. L. Marino, and M. Manic, "ResNet Autoencoders for Unsupervised Feature Learning From High-Dimensional Data: Deep Models Resistant to Performance Degradation," in IEEE Access, vol. 9, DOI: 10.1109/ACCESS.2021.3064819, 2021

7. Vaagensmith B, Kumar Singh V, Ivans R, Marino DL, Wickramasinghe CS, Lehmer J, Phillips T, Rieger C, Manic M., "Review of Design Elements within Power Infrastructure Cyber-Physical Test Beds as Threat Analysis Environments," in Energies 2021, DOI: 10.3390/en14051409, 2021

8. D. Marino, M. Manic, "Modeling and Planning under Uncertainty using Deep Neural Networks", in IEEE Transactions on Industrial Informatics, vol. 15, no. 8, DOI: 10.1109/TII.2019.2917520, 2019

9. D. Marino, J. Tumialan, "Ball and beam control system using Lego NXT", published in the journal: Vision Electronica: algo mas que un estado solido, issue 14, num. 2. ISSN 1909-9746, e-ISSN 2248-4728, 2014.

Peer reviewed conferences:

1. C. Wickramasinghe, D. Marino, and M. Manic, "Deep Embedded Clustering with ResNets," in Proc. 14th International Conference on Human System Interaction, IEEE HSI 2021, Poland, July 8-10. 2021. pp. 1-6, DOI: 10.1109/HSI52170.2021.9538747

2. D. Marino, J. Grandio, C. Wickramasinghe, K. Schroeder, K. Bourne, A.V. Filippas, and M. Manic, "AI Augmentation for Trustworthy AI: Augmented Robot Teleoperation" in Proc. 13th International Conference on Human System Interaction, IEEE HSI 2020, Tokyo, Japan, 2020

3. C. Wickramasinghe, D. Marino, J. Grandio, and M. Manic, "Trustworthy AI Development Guidelines for Human System Interaction," in Proc. 13th International Conference on Human System Interaction, IEEE HSI 2020, Tokyo, Japan, 2020

4. D. Marino, C. Wickramasinghe, C. Rieger, M. Manic, "Data-driven Stochastic Anomaly Detection on Smart-Grid communications using Mixture Poisson Distributions" in Proc. 45th Annual Conference of the IEEE Industrial Electronics Society, IECON, Lisbon, Portugal, 2019

5. D. Marino, Chathurika S. Wickramasinghe, Kasun Amarasinghe, Hari Challa, Philip Richardson, Ananth A. Jillepalli, Brian K. Johnson, Craig Rieger, Milos Manic, "Cyber and Physical Anomaly Detection in Smart-Grids", in Proc. of the IEEE Resilience Week (RW), San Antonio, TX, 2019

6. C. Wickramasinghe, K. Amarasinghe, D. Marino, Z. Spielman, I. Pray, D. Gertman, and M. Manic, "Intelligent Driver System for Improving FuelEfficiency in Vehicle Fleets," in Proc. 12th International Conference on Human System Interaction, IEEE HSI 2019, Richmond VA, USA, June 25-27, 2019

7. M. Stuart, C. Wickramasinghe, D. Marino, D. Kumbhare, K. Holloway, M. Manic, "Machine Learning for Deep Brain Stimulation Efficacy using Dense Array EEG," in Proc. 12th International Conference on Human System Interaction, IEEE HSI 2019, Richmond VA, USA, June 25-27, 2019

8. C. Wickramasinghe, D. Marino, F. Yucel, E. Bulut, and M. Manic, "Data-Driven Hourly Taxi Drop-offs Prediction using TLC Trip Record Data," in Proc. 12th International Conference on Human System Interaction, IEEE HSI 2019, Richmond VA, USA, June 25-27, 2019

9. C. Wickramasinghe, D. Marino, K. Amarasinghe, M. Manic, "Generalization of Deep Learning For Cyber-Physical System Security: A Survey" in Proc. 44th Annual Conference of the IEEE Industrial Electronics Society, IECON 2018, Washington DC, USA, Oct. 21-23, 2018

10. D. Marino, C. Wickramasinghe, M. Manic, "An Adversarial Approach for Explainable AI in Intrusion Detection Systems", in Proc. 44th Annual Conference of the IEEE Industrial Electronics Society, IECON, Washington DC, USA, 2018

11. K. Amarasinghe, C. Wickramasinghe, D. Marino, C.Rieger, M. Manic, "Framework for Data-Driven Health Monitoring of Cyber-Physical Systems" , IEEE Resilience Week (RW) 2018, Denver, CO, USA, Aug 20-23, 2018

12. D. Marino, M. Anderson, K. Kenney, and M. Manic, "Interpretable data-driven modeling in biomass preprocessing", in Proc. 11th International Conference on Human System Interaction, IEEE HSI, Gdansk, Poland, 2018.

13. C. Wickramasinghe, K. Amarasinghe, D. Marino, and M. Manic, "Deep Self-Organizing Maps for Visual Data Mining," in Proc. 11th International Conference on Human System Interaction, IEEE HSI 2018, Gdansk, Poland, July 04-06, 2018

14. D. Marino, K. Amarasinghe, M. Anderson, N. Yancey, Q. Nguyen, K. Kenney, M. Manic , "Data driven decision support for reliable biomass feedstock preprocessing" , IEEE Resilience Week (RWS), Wilmington, DE, USA, 2017.

15. D. Marino, K. Amarasinghe, M. Manic, "Simultaneous Generation-Classification Using LSTM", in IEEE Symposium Series on Computational Intelligence IEEE SSCI, Athens, Greece, 2016.

16. D. Marino, K. Amarasinghe, M. Manic, "Building Energy Load Forecasting using Deep Neural Networks", in Press. 42nd Annual Conference of the IEEE Industrial Electronics Society, IEEE IECON, Florence, Italy, 2016.

17. D. Marino, M. Manic, "Fast Trajectory Simplification Algorithm for Natural User Interfaces in Robot Programming by Demonstration," in Proc. 25th International Symposium on Industrial Electronics, IEEE ISIE, Santa Clara, USA, 2016.

18. D. Marino, J. Tumialan, "Hand Position Tracking Using a Depth Image from a RGB-d Camera", in Proc. IEEE International Conference on Industrial Technology, ICIT, 2015.

19. M. Bueno, D. Marino, "A comparative analysis of adaptive visual servo control for robots manipulators in 2D", in II International Congress on Mechatronics and Automation Engineering (CIIMA), 2013.

Bibliography

[1] D. Marino, C. Wickramasinghe, K. Amarasinghe, H. Challa, P. Richardson, A. Jillepalli, B. Johnson, C. Rieger, and M. Manic, "Cyber and physical anomaly detection insmart-grids," in *IEEE Resilience Week (RW) 2019.* ACM, 2019.

[2] OECD, "OECD Principles on AI." [Online]. Available: https://www.oecd.org/going-digital/ai/principles/

[3] The White House Office of Science and Technology Policy, "American artificial intelligence initiative." [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2020/02/American-AI-Initiative-One-Year-Annual-Report.pdf

[4] I. Saif and B. Ammanath, "'trustworthy ai'is a framework to help manage unique risk," *MIT Technology Review*, 2020.

[5] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild *et al.*, "Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence," USDOE Office of Science (SC), Washington, DC (United States), Tech. Rep., 2019.

[6] D. L. Marino, C. S. Wickramasinghe, V. K. Singh, J. Gentle, C. Rieger, and M. Manic, "The virtualized cyber-physical testbed for machine learning anomaly detection: A wind powered grid case study," *IEEE Access*, 2021.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] D. A. Hashimoto, G. Rosman, D. Rus, and O. R. Meireles, "Artificial intelligence in surgery: promises and perils," *Annals of surgery*, vol. 268, no. 1, p. 70, 2018.

[9] T. G. Rudner and H. Toner, "Key concepts in ai safety: Interpretability in machine learning," *Center for Security and Emerging Technology*, 2021.

[10] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "The robustness of deep networks: A geometrical perspective," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 50–62, 2017.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[12] A. Kwasniewska, J. Ruminski, and M. Szankin, "Improving accuracy of contactless respiratory rate estimation by enhancing thermal sequences with deep neural networks," *Applied Sciences*, vol. 9, no. 20, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4405

[13] K. Czuszynski, J. Ruminski, and A. Kwasniewska, "Gesture recognition with the linear optical sensor and recurrent neural networks," *IEEE Sensors Journal*, vol. 18, pp. 5429 – 5438, 05 2018.

[14] DARPA, "Ai next campaign." [Online]. Available: https://www.darpa.mil/work-with-us/ai-next-campaign

[15] C. S. Wickramasinghe, D. L. Marino, K. Amarasinghe, and M. Manic, "Generalization of deep learning for cyber-physical system security: A survey," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, 2018, pp. 745–751.

[16] C. S. Wickramasinghe, K. Amarasinghe, and M. Manic, "Deep self-organizing maps for unsupervised image classification," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[17] H. A. Abbass, J. Scholz, and D. J. Reid, *Foundations of trusted autonomy*. Springer, 2018, vol. 117.

[18] OCED, "Recommendation of the council on artificial intelligence." [Online]. Available: https://www.fsmb.org/siteassets/artificial-intelligence/pdfs/oecd-recommendation-on-ai-en.pdf

[19] ——, "OCED AI Principals: The Role Of MPS In Leveraging The Benefits Of AI." [Online]. Available: https://www.oecd.org/parliamentarians/meetings/gpn-meeting-october-2019/Dirk-Pilat-OECD-AI-principles-11-Oct-2019.pdf

[20] C. S. Wickramasinghe, D. L. Marino, J. Grandio, and M. Manic, "Trustworthy AI Development Guidelines for Human System Interaction," in *13th International Conference on Human System Interaction (HSI) - in press*, 2020.

[21] High-Level Expert Group on AI, "Ethics guidelines for trustworthy ai," European Commission, Brussels, Report, Apr. 2019. [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai

[22] S. K. Devitt, "Trustworthiness of autonomous systems," in *Foundations of trusted autonomy*. Springer, Cham, 2018, pp. 161–184.

[23] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[24] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[25] D. L. Marino, C. S. Wickramasinghe, and M. Manic, "An adversarial approach for explainable ai in intrusion detection systems," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 3237–3243.

[26] K. Amarasinghe and M. Manic, "Explaining what a neural network has learned: Toward transparent classification," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[27] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.

[28] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA), nd Web*, vol. 2, 2017.

[29] S. Wachter, B. Mittelstadt, and L. Floridi, "Transparent, explainable, and accountable ai for robotics," *Science Robotics*, vol. 2, no. 6, 2017. [Online]. Available: https://robotics.sciencemag.org/content/2/6/eaan6080

[30] S. McConnell, *Code complete*. Pearson Education, 2004.

[31] M. Pezze and M. Young, *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons, 2008.

[32] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, p. 31–57, Jun. 2018. [Online]. Available: https://doi.org/10.1145/3236386.3241340

[33] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf

[34] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (xai) program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, 2019.

[35] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using deep neural networks," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 7046–7051.

[36] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[37] D. Marino, K. Amarasinghe, M. Anderson, N. Yancey, Q. Nguyen, K. Kenney, and M. Manic, "Data driven decision support for reliable biomass feedstock preprocessing," in *2017 Resilience Week (RWS)*, Sep. 2017, pp. 97–102.

[38] D. L. Marino, C. S. Wickramasinghe, K. Amarasinghe, H. Challa, P. Richardson, A. A. Jillepalli, B. K. Johnson, C. Rieger, and M. Manic, "Cyber and physical anomaly detection in smart-grids," in *2019 Resilience Week (RWS)*, vol. 1, 2019, pp. 187–193.

[39] K. Amarasinghe, C. Wickramasinghe, D. Marino, C. Rieger, and M. Manic, "Framework for data driven health monitoring of cyber-physical systems," in *2018 Resilience Week (RW)*, 2018, pp. 25–30.

[40] D. L. Marino, C. S. Wickramasinghe, K. Amarasinghe, H. Challa, P. Richardson, A. A. Jillepalli, B. K. Johnson, C. Rieger, and M. Manic, "Cyber and physical anomaly detection in smart-grids," in *2019 Resilience Week (RWS)*, vol. 1, 2019, pp. 187–193.

[41] N. Science and T. C. U. S. C. on Artificial Intelligence, *The national artificial intelligence research and development strategic plan: 2019 update*. National Science and Technology Council (US), Select Committee on Artificial Intelligence, 2019.

[42] ©[2021] IEEE. Reprinted, with permission from D. L. Marino and M. Manic, "Modeling and planning under uncertainty using deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 8, pp. 4442–4454, 2019.

[43] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," Seattle, USA, June 2009.

[44] ——. (2018) Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation. [Online]. Available: http://underactuated.mit.edu/

[45] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 408–423, Feb 2015.

[46] W. Zine, Z. Makni, E. Monmasson, L. Idkhajine, and B. Condamin, "Interests and limits of machine learning-based neural networks for rotor position estimation in ev traction drives," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 1942–1951, May 2018.

[47] Z. Wang, C. Hu, Y. Zhu, S. He, K. Yang, and M. Zhang, "Neural network learning adaptive robust control of an industrial linear motor-driven stage with disturbance rejection ability," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2172–2183, Oct 2017.

[48] C. Yang, Y. Jiang, Z. Li, W. He, and C. Su, "Neural control of bimanual robots with guaranteed global stability and motion precision," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1162–1171, June 2017.

[49] N. K. Dhar, N. K. Verma, and L. Behera, "Adaptive critic-based event-triggered control for hvac system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 178–188, Jan 2018.

[50] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7559–7566.

[51] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[52] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proc. of the 33nd International Conference on Machine Learning (ICML)*, June 2016, pp. 1050–1059.

[53] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR)*, 2014.

[54] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing, "A benchmark environment motivated by industrial control problems," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov 2017, pp. 1–8.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, May 2015.

[56] Y. Gal, R. T. McAllister, and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," in *ICML Workshop on Data-Efficient Machine Learning*, June 2016.

[57] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[58] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." in *Proc. of the 30th International Conference on Machine Learning (ICML)*, vol. 28, 2013, pp. 1310–1318.

[59] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, May 2010, pp. 249–256.

[60] N. Sun, T. Yang, Y. Fang, B. Lu, and Y. Qian, "Nonlinear motion control of underactuated three-dimensional boom cranes with hardware experiments," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 887–897, March 2018.

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540 [cs.LG]*, June 2016.

[62] E. Coumans and Y. Bai. (2016) Pybullet, a python module for physics simulation for games, robotics and machine learning. [Online]. Available: https://pybullet.org/

[63] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[64] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *arXiv:1707.02286 [cs.AI]*, July 2017.

[65] M. Khodayar, O. Kaynak, and M. E. Khodayar, "Rough deep neural architecture for short-term wind speed forecasting," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 2770–2779, Dec 2017.

[66] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990–5998, July 2018.

[67] H. Hu, B. Tang, X. Gong, W. Wei, and H. Wang, "Intelligent fault diagnosis of the high-speed train with big data based on deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2106–2116, Aug 2017.

[68] W. Lu, B. Liang, Y. Cheng, D. Meng, J. Yang, and T. Zhang, "Deep model based domain adaptation for fault diagnosis," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 3, pp. 2296–2305, March 2017.

[69] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, May 2016.

[70] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[71] F. Guo, H. Kodamana, Y. Zhao, B. Huang, and Y. Ding, "Robust identification of nonlinear errors-in-variables systems with parameter uncertainties using variational bayesian approach," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3047–3057, Dec 2017.

[72] R. Frigola, Y. Chen, and C. E. Rasmussen, "Variational gaussian process state-space models," in *Proc. of the 27th International Conference on Neural Information Processing Systems*, 2014, pp. 3680–3688.

[73] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv:1702.01182 [cs.LG]*, Feb 2017.

[74] M. Al-Shedivat, A. G. Wilson, Y. Saatchi, Z. Hu, and E. P. Xing, "Learning scalable deep kernels with recurrent structure," *Journal of Machine Learning Research*, vol. 18, no. 82, pp. 1–37, 2017.

[75] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," in *5th International Conference on Learning Representations (ICLR)*, April 2017.

[76] T. M. Moerland, J. Broekens, and C. M. Jonker, "Learning multimodal transition dynamics for model-based reinforcement learning," *arXiv:1705.00470 [stat.ML]*, Aug 2017.

[77] ©[2021] IEEE. Reprinted, with permission from D. L. Marino and M. Manic, "Physics enhanced data-driven models with variational gaussian processes," *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 252–265, 2021.

[78] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[79] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018.

[80] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*. Springer, 2004, pp. 27–29.

[81] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 1939–1959, 2005.

[82] M. Titsias, "Variational learning of inducing variables in sparse gaussian processes," in *Artificial Intelligence and Statistics*, 2009, pp. 567–574.

[83] R. Tedrake and the Drake Development Team, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," 2016. [Online]. Available: https://drake.mit.edu

[84] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, 2006, pp. 1257–1264.

[85] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," *arXiv preprint arXiv:1309.6835*, 2013.

[86] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, 2015, pp. 2575–2583.

[87] D. L. Marino and M. Manic, "Modeling and planning under uncertainty using deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 8, pp. 4442–4454, Aug 2019.

[88] B. Donon, B. Donnot, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer, "Leap nets for system identification and application to power systems," *Neurocomputing*, 2020.

[89] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," in *Advances in Neural Information Processing Systems*, vol. 32, 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf

[90] T. Matsubara, A. Ishikawa, and T. Yaguchi, "Deep energy-based modeling of discrete-time physics," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 13 100–13 111.

[91] E. de Bezenac, A. Pajot, and P. Gallinari, "Deep learning for physical processes: Incorporating prior scientific knowledge," *arXiv preprint arXiv:1711.07970*, 2017.

[92] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating eulerian fluid simulation with convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3424–3433.

[93] R. Rai and C. K. Sahu, "Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus," *IEEE Access*, pp. 1–1, 2020.

[94] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[95] Y. Yang and P. Perdikaris, "Adversarial uncertainty quantification in physics-informed neural networks," *Journal of Computational Physics*, vol. 394, pp. 136–152, 2019.

[96] T. Bikmukhametov and J. Jäschke, "Combining machine learning and process engineering physics towards enhanced accuracy and explainability of data-driven models," *Computers & Chemical Engineering*, pp. 106 834–106 861, 2020.

[97] H. Bijl, T. B. Schön, J.-W. van Wingerden, and M. Verhaegen, "System identification through online sparse gaussian process regression with input noise," *IFAC Journal of Systems and Control*, vol. 2, pp. 1–11, 2017.

[98] J. Hall, C. Rasmussen, and J. Maciejowski, "Modelling and control of nonlinear systems using gaussian processes with partial model information," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 5266–5271.

[99] F. M. Gray and M. Schmidt, "A hybrid approach to thermal building modelling using a combination of gaussian processes and grey-box models," *Energy and Buildings*, vol. 165, pp. 56–63, 2018.

[100] M. C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[101] W. Li, S. Chen, Z. Jiang, D. W. Apley, Z. Lu, and W. Chen, "Integrating bayesian calibration, bias correction, and machine learning for the 2014 sandia verification and validation challenge problem," *Journal of Verification, Validation and Uncertainty Quantification*, vol. 1, no. 1, p. 011004, 2016.

[102] T. V. Nguyen and E. V. Bonilla, "Automated variational inference for gaussian process models," in *Advances in Neural Information Processing Systems*, 2014, pp. 1404–1412.

[103] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, "Automatic differentiation variational inference," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 430–474, 2017.

[104] H. Mohammadi, P. Challenor, M. Goodfellow, and D. Williamson, "Emulating computer models with step-discontinuous outputs using gaussian processes," *arXiv preprint arXiv:1903.02071*, 2019.

[105] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth, "Manifold gaussian processes for regression," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 3338–3345.

[106] K. Amarasinghe, C. Wickramasinghe, D. Marino, C. Rieger, and M. Manicl, "Framework for data driven health monitoring of cyber-physical systems," in *2018 Resilience Week (RWS)*, Aug 2018, pp. 25–30.

[107] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Design Automation Conference*, June 2010, pp. 731–736.

[108] A. A. Jillepalli, F. T. Sheldon, D. C. de Leon, M. Haney, and R. K. Abercrombie, "Security management of cyber physical control systems using nist sp 800-82r2," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, June 2017, pp. 1864–1870.

[109] L. Mookiah, C. Dean, and W. Eberle, "Graph-based anomaly detection on smart grid data," in *The Thirtieth International Flairs Conference*, 2017.

[110] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu, "Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 847–855, 2013.

[111] X. Clotet, J. Moyano, and G. León, "A real-time anomaly-based ids for cyber-attack detection at the industrial process level of critical infrastructures," *International Journal of Critical Infrastructure Protection*, vol. 23, pp. 11–20, 2018.

[112] D. L. Marino, C. S. Wickramasinghe, C. Rieger, and M. Manic, "Data-driven stochastic anomaly detection on smart-grid communications using mixture poisson distributions," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 5855–5861, ©2019 IEEE.

[113] Z. Zhao, C. Xu, and B. Li, "A lstm-based anomaly detection model for log analysis," *Journal of Signal Processing Systems*, vol. 93, no. 7, pp. 745–751, 2021.

[114] F. Skopik, M. Landauer, M. Wurzenberger, G. Vormayr, J. Milosevic, J. Fabini, W. Prüggler, O. Kruschitz, B. Widmann, K. Truckenthanner *et al.*, "synergy: Cross-correlation of operational and contextual data to timely detect and mitigate attacks to cyber-physical systems," *Journal of Information Security and Applications*, vol. 54, p. 102544, 2020.

[115] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[116] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in a data-flow environment: Experience in network intrusion detection," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999, pp. 114–124.

[117] C. I. for Cybersecurity, "Nsl-kdd dataset." [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[118] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[119] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[120] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[121] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[122] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[123] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[124] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[125] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[126] Y. Liu, Y. Peng, B. Wang, S. Yao, and Z. Liu, "Review on cyber-physical systems," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 1, pp. 27–40, 2017.

[127] A. Jones, Z. Kong, and C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 848–853.

[128] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, 2017, pp. 140–145.

[129] R. C. Borges Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, "Machine learning for power system disturbance and cyber-attack discrimination," in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*, 2014, pp. 1–8.

[130] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

[131] C. S. Wickramasinghe, D. L. Marino, and M. Manic, "Resnet autoencoders for unsupervised feature learning from high-dimensional data: Deep models resistant to performance degradation," *IEEE Access*, vol. 9, pp. 40 511–40 520, 2021.

[132] D. L. Marino, C. S. Wickramasinghe, B. Tsouvalas, C. Rieger, and M. Manic, "Data-driven correlation of cyber and physical anomalies for holistic system health monitoring," in *IEEE ACCESS*. IEEE, 2021, ©2021 IEEE.

[133] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt *et al.*, "Support vector method for novelty detection." in *NIPS*, vol. 12. Citeseer, 1999, pp. 582–588.

[134] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[135] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ser. ODD '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 8–15. [Online]. Available: https://doi.org/10.1145/2500853.2500857

[136] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320316300267

[137] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class svm for anomaly detection," in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, vol. 5. IEEE, 2003, pp. 3077–3081.

[138] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, 2013, pp. 8–15.

[139] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems," in *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 2006, pp. 488–498.

[140] S. Teng, N. Wu, H. Zhu, L. Teng, and W. Zhang, "Svm-dt-based adaptive and collaborative intrusion detection," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 108–118, 2018.

[141] M. Alshawabkeh, B. Jang, and D. Kaeli, "Accelerating the local outlier factor algorithm on a gpu for intrusion detection systems," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3. New York, NY, USA: Association for Computing Machinery, 2010, p. 104–110. [Online]. Available: https://doi.org/10.1145/1735688.1735707

[142] W. Wang and P. Lu, "An efficient switching median filter based on local outlier factor," *IEEE Signal Processing Letters*, vol. 18, no. 10, pp. 551–554, 2011.

[143] R. Sandhya, J. Prakash, and B. V. Kumar, "Comparative analysis of clustering techniques in anomaly detection wind turbine data." *Journal of Xi'an University of Architecture & Technology*, vol. 12, no. 3, pp. 5684–5694, 2020.

[144] M. Ahmed, A. Anwar, A. N. Mahmood, Z. Shah, and M. J. Maher, "An investigation of performance analysis of anomaly detection techniques for big data in scada systems." *EAI Endorsed Trans. Indust. Netw. & Intellig. Syst.*, vol. 2, no. 3, p. e5, 2015.

[145] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[146] C. S. Wickramasinghe, D. L. Marino, K. Amarasinghe, and M. Manic, "Generalization of deep learning for cyber-physical system security: A survey," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 745–751.

[147] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 665–674.

[148] K. Sohn, C.-L. Li, J. Yoon, M. Jin, and T. Pfister, "Learning and evaluating representations for deep one-class classification," *arXiv preprint arXiv:2011.02578*, 2020.

[149] C.-L. Li, K. Sohn, J. Yoon, and T. Pfister, "Cutpaste: Self-supervised learning for anomaly detection and localization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9664–9674.

[150] M.-I. Georgescu, A. Barbalau, R. T. Ionescu, F. S. Khan, M. Popescu, and M. Shah, "Anomaly detection in video via self-supervised and multi-task learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 742–12 752.

[151] N. Komodakis and S. Gidaris, "Unsupervised representation learning by predicting image rotations," in *International Conference on Learning Representations (ICLR)*, 2018.

[152] I. Golan and R. El-Yaniv, "Deep anomaly detection using geometric transformations," *arXiv preprint arXiv:1805.10917*, 2018.

[153] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," *arXiv preprint arXiv:1906.12340*, 2019.

[154] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[155] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[156] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," in *Proceedings of the conference on fairness, accountability, and transparency*, 2019, pp. 220–229.

[157] I. Osband, Z. Wen, M. Asghari, M. Ibrahimi, X. Lu, and B. Van Roy, "Epistemic neural networks," *arXiv preprint arXiv:2107.08924*, 2021.

[158] J. Mukhoti, A. Kirsch, J. van Amersfoort, P. H. Torr, and Y. Gal, "Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty," *arXiv preprint arXiv:2102.11582*, 2021.

[159] A. G. de G. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani, "Gaussian process behaviour in wide deep neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1-nGgWC-

[160] R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Bayesian deep convolutional networks with many channels are gaussian processes," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=B1g30j0qF7

[161] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, "On exact computation with an infinitely wide neural net," *arXiv preprint arXiv:1904.11955*, 2019.

[162] L. Noci, G. Bachmann, K. Roth, S. Nowozin, and T. Hofmann, "Precise characterization of the prior predictive distribution of deep relu networks," *arXiv preprint arXiv:2106.06615*, 2021.

[163] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *arXiv preprint arXiv:1612.01474*, 2016.

[164] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[165] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 4759–4770.

[166] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[167] J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan, "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness," *arXiv preprint arXiv:2006.10108*, 2020.

[168] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz, "Neural tangents: Fast and easy infinite neural networks in python," *arXiv preprint arXiv:1912.02803*, 2019.

[169] J. H. Huggins, T. Campbell, M. Kasprzak, and T. Broderick, "Scalable gaussian process inference with finite-data mean and variance guarantees," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 796–805.

[170] T. M. Thomas and A. J. Pollard, "Vaccine communication in a digital society," *Nature materials*, vol. 19, no. 4, pp. 476–476, 2020.

[171] B. Fischhoff, "Communicating uncertainty fulfilling the duty to inform," *Issues in Science and Technology*, vol. 28, no. 4, pp. 63–70, 2012.

[172] D. Kahneman, O. Sibony, and C. R. Sunstein, *Noise: a flaw in human judgment*. Little, Brown, 2021.

[173] Y.-H. Lee, N. E. Dunbar, C. H. Miller, B. L. Lane, M. L. Jensen, E. Bessarabova, J. K. Burgoon, B. J. Adame, J. J. Valacich, E. A. Adame, E. Bostwick, C. W. Piercy, J. Elizondo, and S. N. Wilson, "Training anchoring and representativeness bias mitigation through a digital game," *Simulation & Gaming*, vol. 47, no. 6, pp. 751–779, 2016. [Online]. Available: https://doi.org/10.1177/1046878116662955

[174] C. Smith, R. Youngblood, C. Everett, T. Miyake, M. Mankosa, C. Frepoli, and R. Sampath, *Treatment of uncertainties for security-related design aspects of advanced reactors when using a risk-informed licensing approach*. INL, 2021.

[175] A. J. McHutchon, "Nonlinear modelling and control using gaussian processes," Ph.D. dissertation, Citeseer, 2015.

[176] A. C. Damianou, M. K. Titsias, and N. D. Lawrence, "Variational inference for latent variables and uncertain inputs in gaussian processes," *Journal of Machine Learning Research*, vol. 17, no. 42, pp. 1–62, 2016. [Online]. Available: http://jmlr.org/papers/v17/damianou16a.html

[177] R. McAllister and C. E. Rasmussen, "Data-efficient reinforcement learning in continuous state-action gaussian-pomdps," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/5eac43aceba42c8757b54003a58277b5-Paper.pdf

[178] OECD, *OECD Guidelines on Measuring Trust*, 2017. [Online]. Available: https://www.oecd-ilibrary.org/content/publication/9789264278219-en

[179] M. Brzowski and D. Nathan-Roberts, "Trust measurement in human–automation interaction: A systematic review," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 63, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2019, pp. 1595–1599.

[180] E. L. Glaeser, D. I. Laibson, J. A. Scheinkman, and C. L. Soutter, "Measuring trust," *The quarterly journal of economics*, vol. 115, no. 3, pp. 811–846, 2000.

[181] C. S. Wickramasinghe, D. L. Marino, J. Grandio, and M. Manic, "Trustworthy ai development guidelines for human system interaction," in *2020 13th International Conference on Human System Interaction (HSI)*. IEEE, 2020, pp. 130–136.

[182] R. Rodriguez-Torrado, P. Ruiz, L. Cueto-Felgueroso, M. C. Green, T. Friesen, S. Matringe, and J. Togelius, "Physics-informed attention-based neural network for solving non-linear partial differential equations," *arXiv preprint arXiv:2105.07898*, 2021.

[183] L. Yang, X. Meng, and G. E. Karniadakis, "B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data," *Journal of Computational Physics*, vol. 425, p. 109913, 2021.

[184] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[185] R. Dandekar, K. Chung, V. Dixit, M. Tarek, A. Garcia-Valadez, K. V. Vemula, and C. Rackauckas, "Bayesian neural ordinary differential equations," *arXiv preprint arXiv:2012.07244*, 2020.

[186] G. Yang, "Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes," *33rd Conference on Neural Information Processing Systems*, 2019.

[187] J. Hron, Y. Bahri, J. Sohl-Dickstein, and R. Novak, "Infinite attention: Nngp and ntk for deep attention networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4376–4386.

[188] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4470–4479.

[189] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, "Discovering symbolic models from deep learning with inductive biases," *arXiv preprint arXiv:2006.11287*, 2020.

[190] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[191] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1868447.1868466

[192] J. Edge, "Preparing for the realtime future." [Online]. Available: https://lwn.net/Articles/830660/

[193] P. ACRN, "Project ACRN: A Big Little Hypervisor for IoT Development." [Online]. Available: https://projectacrn.org/

[194] K. B. Petersen and M. S. Pedersen. (2008) The matrix cookbook. [Online]. Available: https://www.math.uwaterloo.ca/%7Ehwolkowi/matrixcookbook.pdf

Vita

Daniel L Marino was born on January 7, 1992 in Bogota, Colombia. He graduated from Gimnasio Fidel Cano, Bogota, Colombia in 2009. He received his B.Eng. in automation engineering from La Salle University, Colombia, in 2015. He has over six years of research and development experience, collaborating with US DOE National Labs, universities, and industry partners. He has authored over 27 articles in peer reviewed journals and conferences. He received the IEEE IES student paper travel award in 2016 and 2019, the VCU CS Outstanding Paper Award in 2020, the VCU CS outstanding early-career student researcher award in 2017, and the honor scholarship granted by La Salle University from 2010 to 2013. His research interests include stochastic modeling, deep learning, and explainable AI with applications in cyber-physical systems, energy, and robotics.