Classification Based Diagnosis: Integrating Partial Knowledge of the Physical System

Ion Matei[1], Johan de Kleer[2], Alexander Feldman[3], Maksym Zhenirovskyy[4], and Rahul Rai[5]

[1,2,3,4] *Palo Alto Research Center, Palo Alto, CA, 94304, USA*
imatei@parc.com
dekleer@parc.com
afeldman@parc.com
mazhenir@parc.com
[5] *University at Buffalo - SUNY, Buffalo, NY, 14260, USA*
rahulrai@buffalo.edu

ABSTRACT

Machine learning methods based on classifiers are more robust to system complexity, but they ignore the relations that exist in the data due to the physical laws governing the behavior of the system. In this paper we discuss how (partial) knowledge about the physical system can be integrated in the machine learning process. We focus on classification based diagnosis. We show how the partially known model is integrated in the classification algorithm, and how the new algorithm differs from the typical classification algorithm used in machine learning. We demonstrate that by integrating the partial system knowledge, the cross-entropy optimization problem used for learning a classifier can be expressed as a set of regression problems in terms of the parameters of the model representing the unknown behavior, followed by simpler classifier learning. The regression problems have reduced complexity since they have to model on a part of the system behavior. We showcase our approach when diagnosis faults for a rail switch system. Our approach amounts to deriving a formal approach to integrating partial system knowledge in classification-based diagnosis.

1. INTRODUCTION

Machine learning algorithms are a useful tool for system analytics applications such as diagnosis and prognostics. They are robust to system complexity but agnostic to the source of the training data. In addition, they typically require more complex models, e.g., neural networks (NN) with many layers. In many applications, we do have at least some partial knowledge about the system from which the training data originates. In several of our previous commercial projects we encountered exactly this case: we had access to the specifications of only a subset of the system components due to proprietary reasons (Matei, Ganguli, Honda, & de Kleer, 2015). If full information about the system is available, a plethora of model-based methods for diagnosis and prognosistics can be used (de Kleer, Mackworth, & Reiter, 1992),(Gertler, 1998),(Isermann, 2005),(Patton, Frank, & Clark, 2000). These methods require some prior information about the fault rates, do not always scale with the system complexity and they work well for particular classes of systems. For example, Kalman filter-based methods (Kalman, 1960) are optimal for linear systems with Gaussian noise. Machine learning methods based on classifiers are more robust to system complexity, but they ignore the relations that exist in the data due to the physical laws governing the behavior of the system.

In this paper we discuss how (partial) knowledge about the system can be integrated in the classifier learning process. We focus on classification problems as they are suitable for diagnosis purposes. We address two main challenges: (i) representation and integration of the unknown behavior, and (ii) design of a training algorithm that considers the partial system knowledge. To address these challenges we build upon our previous work on learning acausal components in partially known physical systems (Matei, de Kleer, & Minhas, 2018). Unlike causal systems, their components do not have a fixed notion of inputs and outputs. They are characterized by ports through which energy is exchanged between components. Component behaviors are described by constitutive equations in terms of port and internal variables. The system behavior emerges from the composition of individual component behaviors through port connections. Acausal systems are typically represented as differential algebraic equations (DAEs). Under certain conditions, by employing index reduction techniques, they can be transformed into ordinary dif-

ferential equations (ODEs) and solved using standard ODE solvers.

We address the first challenge (i) by bringing the system of equations into a block-lower-triangular (BLT) form. This form describes the causal relations between component variables: the equations from which variables are computed, and what variables need to be computed first in order to compute other variables. Hence, we can derive an input-output representation (e.g., a regression model, or a recurrent NN) to represent the unknown behavior, and more importantly we can compose this representation with the rest of the known components. The second challenge (ii) is addressed by showing that a cross-entropy optimization problem used for learning a classifier can be expressed as a set of regression problems in terms of the parameters of the model representing the unknown behavior, followed by a simpler classifier learning. Our approach amounts to deriving a formal approach to integrating partial system knowledge in classification-based diagnosis.

Paper structure: Section 2 describes the acausal model representation, the implications of a partially known model and the classification problem formulation. Section 3 presents the representation of the classifier in the full and partial model knowledge. Section 4 shows an illustrative example.

2. PROBLEM SETUP

Our objective is to diagnose faults in a physical system. The nominal and the fault behaviors represent different operation modes. By diagnosing a fault we mean identifying an operation mode.

2.1. Model representation

We assume that the behavior of the physical system is described by a hybrid differential algebraic equation (DAE). The typical mathematical model for describing the behavior of the system is

$$0 = F(\dot{x}, x, u, w, \theta), \quad (1)$$
$$y = h(x, u, v, \theta), \quad (2)$$

where x is the state vector, u is the vector of inputs, and w and v are process and measurement noise, respectively. The system output is denoted by y and θ is a variable that sets the mode of operation and takes values in the discrete set $\{1, 2, \ldots, M\}$. It is sometimes more beneficial to work with discrete dynamics of the form

$$0 = F(x_{k+1}, x_k, u_k, w_k, \theta_k), \quad (3)$$
$$y_k = h(x_k, u_k, v_k, \theta_k), \quad (4)$$

which can be obtained through approximations of the continuous dynamics, e.g., by approximating the state derivatives.

An example of an electric circuit whose behavior is described by a hybrid DAE is shown in Figure 1. The circuit has two

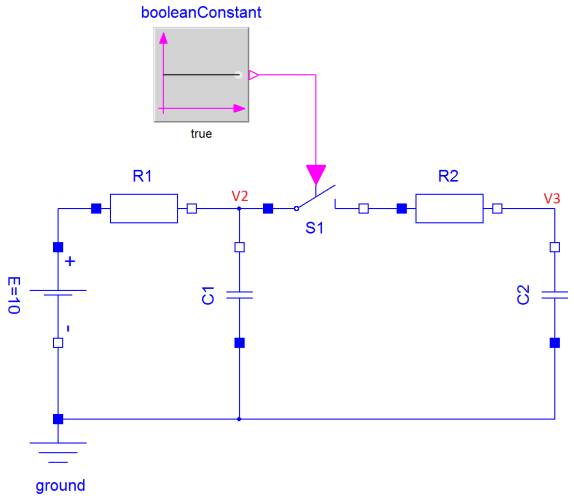

Figure 1. Example of a physical system with a hybrid DAE

modes of operations that are activated by manipulating an electric switch. In the nominal mode the switch is closed. An open switch models an open connection for the resistor R2. In the nominal mode, the behavior of the system is given by

$$u_{R_1} = E + v_2 \quad (5)$$
$$i_E = -\frac{1}{R_1} u_{R_1} \quad (6)$$
$$u_{R_2} = v_3 - v_2 \quad (7)$$
$$i_{C_2} = \frac{1}{R_2} u_{R_2} \quad (8)$$
$$i_{C_1} = i_E - i_{C_2} \quad (9)$$
$$\frac{d}{dt} v_2 = \frac{1}{C_1} i_{C_1} \quad (10)$$
$$\frac{d}{dt} v_3 = \frac{1}{C_2} i_{C_2} \quad (11)$$

while in the fault mode we have

$$u_{R_1} = E + v_2 \quad (12)$$
$$i_{C_1} = -\frac{1}{R_1} u_{R_1} \quad (13)$$
$$\frac{d}{dt} v_2 = \frac{1}{C_1} i_{C_1} \quad (14)$$
$$(15)$$

Note that the two set of equations describe two DAEs. However, by simple substitutions they can be converted into ODEs. For example the fault mode equation takes the form

$$\frac{d}{dt} v_2 = -\frac{1}{R_1 C_1} v_2 - \frac{1}{R_1 C_1} E. \quad (16)$$

We often prefer to preserve the algebraic equations as they can give us key insights into the behavior of specific components.

2.2. Partially known behavior

In an ideal case, we know both the topological and behavioral representation of the system. In real scenarios however, this is rarely the case as we have discovered in one of our previous diagnosis projects described in (Matei et al., 2015). One common cause for lacking full system description is incomplete technical specifications: often, even the system manufacturers do not have access to the complete list of component specifications due to proprietary reasons. In the context of this paper, partial knowledge refers to having access to the behavioral description of a subset of the system components. We do assume that the topological description of the system is known. To make it more concrete, let the behavior of resistor R2 be unknown. How to chose acausal mathematical component models was discussed in (Matei et al., 2018). The component model must contain two connectors, each connector having a current and a potential variable. For our example these are v_2, i_2, v_3 and i_3, where the indices refer to the nodes 2 and 3 in the circuit shown in Figure 1. These variables are constrained by a vector valued function $f_{R_2} : \mathbb{R}^4 \to \mathbb{R}^2$, such that $f_{R_2}(v_2, i_2, v_3, i_3; \beta) = 0$, where β is a set of unknown model parameters. To simplify the model we can assume that $i_2 + i_3 = 0$. Therefore we are left with finding a function $f'_{R_2} : \mathbb{R}^3 \to \mathbb{R}$ such that $f'_{R_2}(v_2, i_2, v_3; \beta) = 0$. This is not a causal representation. We further obtain a causal representation by leveraging the BLT form of the circuit shown in Figure 2, where we assumed some mockup constitutive equation for R2 for the purpose of performing the transform. The BLT form shows that the capacitor's potentials can be interpreted as inputs for the resistor model and the current is an output. This input-output mapping is particular to this circuit though. Therefore, the causal representation for the behavior of the circuit can be chosen as $i_2 = f''_{R_2}(v_2, v_3; \beta)$. There is no systematic way to chose a particular representation. We can chose a polynomial or a NN representation. The causal block representation has the advantage that enables us to model the behavior of the unknown component using an input-output map (or an ODE with inputs and outputs). This map is parameterized by β and the parameters are learned using training data. It has one important disadvantage though. The causal model for the unknown component is not necessarily generalizable. The reason is that the behavior of the component is not actually causal. In other configurations, the component may have a different causal representation, that is, the current may act as input and the potentials as outputs. There is an additional challenge caused by the parameters of the map. Not all of them are feasible. For example, the resistance value is always positive. A negative value will result in an unstable system. Feasibility constraints can also be learned as discussed in (Matei et al., 2018) or derived from component properties such as dissipativity[1]. Alternatively, we can just ignore the existence of constraints and perform unconstrained optimization since we expect the cost function to increase significantly for unstable cases.

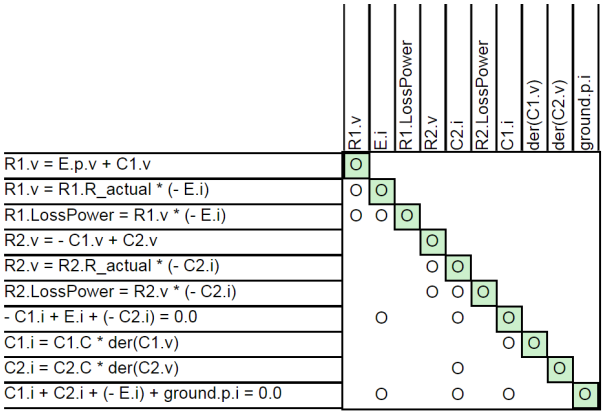

Figure 2. BLT form of the circuit shown in Figure 1. The columns show model variables and the row depict the equations from which the variables are computed. The equations and variables are expressed in the Modelica language

2.3. Classification problem

In the classification problem, the objective is to determine the mode θ based on a set of observations. Without loss of generality, we assume that system has no exogenous inputs.

The type of observations we consider are time series of output measurements $y_{0:T} = \{y_0, y_1, \ldots, y_T\}$, where $y_k = y(t_k)$ and t_k are sampling instants, assumed uniform. To simplify the notation we will generically denote a sequence $y_{0:T}$ by $\boldsymbol{y}$. We will distinguish between time series sample by using the index i, that is $\boldsymbol{y}^{(i)}$. We will make the following assumption.

Assumption 2.1 *The mode θ does not change for the duration $[0, T]$ and all time series correspond to the same initial condition.* □

This means that each data sequence $\boldsymbol{y}^{(i)}$ corresponds to one mode only. The classification problem involves determining the current mode of operation based on a set of observation $\boldsymbol{y}$. It is based on a probabilistic model $p(\theta = n|\boldsymbol{Y} = \boldsymbol{y}; \beta)$, where $\boldsymbol{Y}$ is a vector-valued random variable representing the observations (feature vector). The vector β represents the parameters corresponding to the model used for describing the conditional probability distribution. For example, we can use a NN model with a softmax function at the last layer. The classification decision is the solution of the problem $\arg\max_j \{p(\theta = j|\boldsymbol{Y} = \boldsymbol{y})\}$. The parameters β are learned

[1]The dissipativity constraint for component R_2 requires the power $P = i_2 v_2 + i_3 v_3 \geq 0$. For a resistor model, this means that $P = i_2(v_2 - v_3) = R_2 i_2^2$. This shows that we require $R_2 \geq 0$.

by minimizing the cross-entropy between two probability distributions

$$\min_{\beta} E\left[H\left(q(\theta|\boldsymbol{Y}), p(\theta|\boldsymbol{Y};\beta)\right)\right], \tag{17}$$

where H is the cross entropy defined as $H(q,p) = -E_q[\log(p)]$, and the probability distribution $q(\theta|\boldsymbol{y})$ is the "ground truth", assumed known. To evaluate the expectation (17) we need the unknown distribution of $\boldsymbol{Y}$. This distribution is approximated using the training examples, resulting in

$$E\left[H(q(\theta|\boldsymbol{Y}), p(\theta|\boldsymbol{Y};\beta)\right] \approx$$

$$\frac{1}{N}\sum_{i=1}^{N} H\left(q(\theta|\boldsymbol{Y}=\boldsymbol{y}^{(i)}), p(\theta|\boldsymbol{Y}=\boldsymbol{y}^{(i)};\beta)\right), \tag{18}$$

where $\{\boldsymbol{y}^{(i)}\}_{i=1}^{N}$ is a set of realizations of $\boldsymbol{Y}$ (training examples). The cross-entropy can be explicitly written as

$$H\left(q(\theta|\boldsymbol{y}^{(i)}), p(\theta|\boldsymbol{y}^{(i)};\beta)\right) =$$

$$-\sum_{j=1}^{M} q(\theta=j|\boldsymbol{Y}=\boldsymbol{y}^{(i)}) \log p(\theta=j|\boldsymbol{Y}=\boldsymbol{y}^{(i)}), \tag{19}$$

where $q(\theta=j|\boldsymbol{Y}=\boldsymbol{y}^{(i)})=1$ if $\boldsymbol{y}^{(i)}$ corresponds to mode j, and zero otherwise. In the machine learning community, the solution of Eq. (17) is typically obtained by using gradient descent algorithms, e.g., stochastic gradient descent, Adams (Kingma & Ba, 2014) or RMSProp (Ruder, 2016). The learning algorithm does not use any information about the origin of the data, or what information may be known about the system that generated it. One immediate consequence is that we may require a complex model for $p(\theta|\boldsymbol{Y};\beta)$, and hence a large number of parameters to learn. This in turn induces the need for large training data sets. Another consequence is that we ignore relations that exist between the elements of the feature vectors. Such relations originate from the physical laws governing the behavior of the physical system.

3. CLASSIFIER TRAINING THAT INCLUDES INFORMATION ABOUT THE SYSTEM

We distinguish two cases concerning what is known about the system generating the observations: (i) complete knowledge, and (ii) partial knowledge.

3.1. Complete knowledge

In this scenario, a complete model of the physical system is available. This model accurately describes the behavior of the system up to some process and measurement noises. The objective is to find a representation of the probability $p(\theta|\boldsymbol{y})$. Using Bayes's rule, this probability can be expressed as

$$p(\theta=j|\boldsymbol{Y}=\boldsymbol{y}) =$$

$$\frac{p(\boldsymbol{y}|\theta=j)p(\theta=j)}{\sum_{l=1}^{M} p(\boldsymbol{y}|\theta=l)p(\theta=l)}. \tag{20}$$

The computation of the probability $p(\theta=j|\boldsymbol{Y}=\boldsymbol{y})$ can be done using the model of the system. Using the discrete dynamics shown in Eq. (3)-(4), we have

$$p(y_{0:T}|\theta=j) =$$

$$\int p(y_T|x_T,\theta=j)p(x_T|y_{0:T-1},\theta=j)dx_T. \tag{21}$$

The probability $p(y_T|x_T,\theta=j)$ is completely determined by the sensing model described by Eq. (4) and the distribution of the measurement noise v_T. In the case v_T is an additive Gaussian noise, $p(y_T|x_T,\theta=j)$ is a Gaussian probability distribution function (pdf). The quantity $p(x_T|y_{0:T-1},\theta=j)$ is the prediction step in the state estimation procedure. It can be expressed in terms of the update step:

$$p(x_T|y_{0:T-1},\theta=j) =$$

$$\int p(x_T|x_{T-1},\theta=j)p(x_{T-1}|y_{0:T-1}\theta=j)dx_{T-1}. \tag{22}$$

The probability $p(x_T|x_{T-1},\theta=j)$ is determined by the process model defined by Eq. (3) and by the distribution of the process noise w_{T-1}. The probability $p(x_{T-1}|y_{0:T-1},\theta=j)$ is the update step in the state estimation process. Therefore, we require M state estimation filters run in parallel, for each mode of operation. The complexity of evaluating the iterative convolution operations involving the probabilities at the prediction and update steps depend on the type of model. For linear systems with Gaussian noise these probabilities are Gaussian with statistics computed using the Kalman filter (Kalman, 1960) equations. For nonlinear systems, extensions of the Kalman filter such as the extended or unscented Kalman filter may be an option. Alternatively, provided sufficient computational resources are made available, we can use the particle filter (Arulampalam, Maskell, & Gordon, 2002). If unknown, the probability $p(\theta=j)$ can compute as proxy using the training examples. Namely, we have $p(\theta=j) \approx \frac{1}{N}\sum_{i=1}^{N} q(\theta=j|\boldsymbol{y}^{(i)})$. Alternatively, we can solve an optimization problem of the form defined by expression (17) with respect to the probabilities $p(\theta=j)$. We can model these probability using a softmax function, $p(\theta=j) = \frac{e^{\eta_j}}{\sum_{l=1}^{M} e^{\eta_l}}$, and solve the optimization problem with respect to parameters η_j.

Figure 3 depicts the architecture of the classifier when the complete model of the system is known. It is composed of M filters that compute the probability distribution of the outputs given a mode of operation, followed by a fusion block, which determines the current model by computing the probability introduced in Eq. (20).

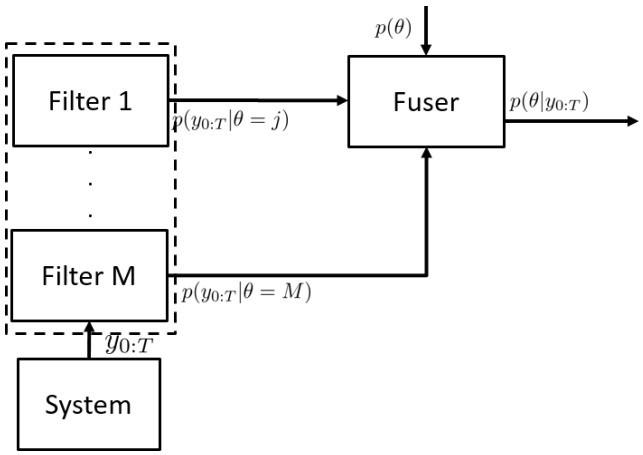

Figure 3. Classifier architecture: complete knowledge case

3.2. Partial knowledge

In the partial model knowledge case, the classification problem uses the same formula as in Eq. (20), and hence we need to evaluate the probabilities $p(\boldsymbol{y}|\theta = j)$ and $p(\theta = j)$ if unknown. To evaluate $p(\boldsymbol{y}|\theta = j)$ we need the complete model, which at this point is not available, since the parameterized maps modeling the unknown components are not tuned to match the observed behavior. One alternative is to augment the state of the system with the parameters of the unknown components and learn them as part of the state estimation problems. We would need to use a filter that is accurate enough for non-linear systems since the parameters may enter non-linearly in the behavioral equations. Another approach is to use an optimization based approach for learning the parameters. This fits more naturally with learning classifiers in machine learning approaches. We consider two strategies for learning the parameters of the unknown components and switching model. In the first strategy, we first learn separately the parameters of the unknown part of the system, followed by learning the switching model $p(\theta = j)$. In the second strategy we jointly learn the unknown component parameters and the switching parameters. The application of one or the other depends on particular assumptions we make. For both strategies we model the probability distribution $p(y_{0:T}|\theta = j)$ as a Gaussian multivariate distribution with unknown covariance matrix. Assuming independent, additive measurement noise, it is formally expressed as

$$p(\boldsymbol{y}|\theta = j; \beta_j) \approx p(\boldsymbol{y}|\hat{\boldsymbol{y}}, \theta = j; \beta_j) = \prod_{i=n}^{m} p(\hat{y}_i + v_i|\theta = j; \beta_j), \tag{23}$$

where $p(y_i|\theta_j, \hat{y}_j; \beta_j) = p(\hat{y}_i + v_i|\theta_j; \beta_j) \sim \mathcal{N}(\hat{y}_i, \Sigma_j)$, Σ_j is the noise covariance matrix, and $\hat{y}_i$ is an entry in the simulated output sequence $\hat{y}_{0:T}$ using the model in mode j; model that dependents on the unknown vector of parameters β_j. If the parameters of the component are mode independent, we can use the same parameters for each mode.

3.2.1. Sequential parameter learning

We make the assumption that the process noise is negligible and that the variance of the measurement noise is unknown. The variance will be estimated as part of the learning process. For each mode j, the parameters β_j are learned by solving a minimum least square error problem of the form

$$\min_{\beta_j} \frac{1}{N_j} \sum_{i=1}^{N_j} \|y_{0:T}^{(i)} - \hat{y}_{0:T}^{(i)}(\beta_j)\|^2, \tag{24}$$

where index i refers to a training example, and N_j is the number of training examples corresponding to mode j. Any non-linear least square optimization algorithm can be used, the numerical complexity coming from the fact that the optimization algorithm requires simulating the model at each iteration and computing the gradient of the cost function, if a gradient-based algorithm is used. To obtain analytic formulas for the gradient of the cost function, we can use the auto-differentiation feature of deep learning platforms such as Tensorflow (Abadi et al., 2015), Pytorch (Subramanian, 2018), or Autograd (Maclaurin, Duvenaud, Johnson, & Adams, 2015). All three option support loss functions that can depend on ODE solutions. They do not support DAEs though for which a causal graph representation of the gradient computation scheme is not suitable. An alternative to automatic differentiation is using DAE solvers that support sensitivity analysis (e.g., CVODES, IDAS). An example of a Python package that implements DAE solvers featuring sensitivity analysis is DAETools (Nikolic, 2016a), where sensitivities of the DAE variables with respect to the system parameters can be computed numerically, but accurately at the same time with the DAE solution. Formulating the system dynamics in a deep-learning framework enables the use of GPUs that can prove beneficial for large scale problems and for large training data sets. Once the optimization problem is executed, we can use the empirical covariance as an approximation for Σ_j, namely

$$\Sigma_j \approx \frac{1}{N_j(T+1)} \sum_{i=1}^{N_j} \sum_{l=0}^{T} \left[y_l^{(i)} - \hat{y}_l^{(i)}\right] \left[y_l^{(i)} - \hat{y}_l^{(i)}\right]', \tag{25}$$

where $\hat{y}_l^{(i)}$ are functions of β_j^*, the optimal parameters as produces by the optimization problem. Next, we compute the probabilities $p(\theta = j; \eta)$. For this part we follow the same idea as in Section 3.1, where we solve an optimization problem shown in expression (17), in terms of a parameterized model $p(\theta = j; \eta)$. The sequential learning algorithm is summarized in Algorithm 1. One drawback of this approach is that any errors accumulated while learning parameters β_j will affect the mode switching part of the algorithm. We address this in the second strategy.

Algorithm 1 Sequential learning

1: **for** $j = 1 : M$ **do**
2: Learn **offline** the parameters for the unknown components in mode j

$$\beta_j^* = \arg\min_{\beta_j} \sum_i \|y_{0:T}^{(i)} - \hat{y}_{0:T}^{(i)}\|^2$$

3: Estimate the error covariance matrix covariance matrix

$$\Sigma_j \approx \frac{1}{N_j(T+1)} \sum_{i=1}^{N_j} \sum_{l=0}^{m} \left[y_l^{(i)} - \hat{y}_l^{(i)}\right] \left[y_l^{(i)} - \hat{y}_l^{(i)}\right]',$$

4: Estimate the switching parameter

$$\eta^* = \arg\min_{\eta} - \sum_i \sum_{j=1}^{M} q(\theta = j|y_{0:T}^{(i)}) \log p(\theta = j|y_{0:T}^{(i)}; \beta_j^*, \eta)$$

subject to:

$$p(\theta = j|y_{0:T}; \beta_j^*, \eta) = \frac{p(y_{0:T}|\theta = j; \beta_j^*)\eta_j}{\sum_{l=1}^{M} p(y_{0:T}|\theta = l; \beta_l^*)\eta_l}$$

where

$$p(y_{0:T}|\theta = j; \beta_j^*) \sim \mathcal{N}(\hat{y}_{0:T}, \Sigma_j),$$

$$\eta_j \geq 0, \sum_{j=1}^{M} \eta_j = 1,$$

with $\eta = [\eta_1, \ldots, \eta_M]$, $\hat{y}_{0:T}$ the simulated outputs in mode j.
5: Predict **online** the mode

$$j^* = \arg\max_j p(\theta = j|y_{0:T}; \beta_j^*, \eta^*)$$

3.2.2. Joint parameter learning

Here we assume that the variance of the measurement noise Σ_j is known and hence we do not need to estimate it. This is the case when the precision of the sensors is available. We maintain the same assumptions on the sensing model that induces a Gaussian distribution for the random vector $y_{0:T}|\theta = j$. This way we can formulate an optimization problem where both the parameters of the unknown components and the switching parameters can be estimated simultaneously. This classification approach is summarized in Algorithm 2. As usual with non-convex optimization problems, convergence to the global minima is not guaranteed. Still, it is usually the case that the cost function has a rich set of local minima that provide satisfactory prediction accuracy.

4. ILLUSTRATIVE EXAMPLE

To showcase our approach, we develop a diagnosis engine for detecting and isolating faults in a rail switch system. We consider a set of faults for which we build a hybrid classifier that uses that partial system knowledge and a NN-based classifier, for comparison purposes.

Algorithm 2 Joint learning

1: Solve **offline** the optimization problem

$$\beta_j^*, \eta^* = \arg\min_{\beta_j, \eta} - \sum_i \sum_{j=1}^{M} q(\theta = j|y_{0:T}^{(i)}) \log p(\theta = j|y_{0:T}^{(i)}; \beta_j, \eta)$$

subject to:

$$\eta_j \geq 0, \sum_{j=1}^{M} \eta_j = 1,$$

$$p(\theta = j|y_{0:T}; \beta_j, \eta) = \frac{p(y_{0:T}|\theta = j; \beta_j)\eta_j}{\sum_{l=1}^{M} p(y_{0:T}|\theta = l; \beta_l)\eta_l}$$

where

$$p(y_{0:T}|\theta = j; \beta_j) = \prod_{i=0}^{T} p(\hat{y}_i + v_i|\theta = j; \beta_j)$$

and $p(\hat{y}_i + v_i) \sim \mathcal{N}(\hat{y}_i, \Sigma_j)$, with $\hat{y}_{0:T}$ the simulated sequence of outputs in mode j, and Σ_j the measurement noise covariance matrix.
2: Predict **online** the mode

$$j^* = \arg\max_j p(\theta = j|y_{0:T}; \beta_j^*, \eta^*)$$

4.1. Rail switch model description

The rail switch is composed of a servo-motor and a gear-mechanism for scaling the rotational motion and for amplifying the torque generated by the electrical motor. The rail load is composed by a mechanical adjuster, and tongue-rails. The schematics of the system is presented in Figure 4 depicting the main components of the rail switch. The point

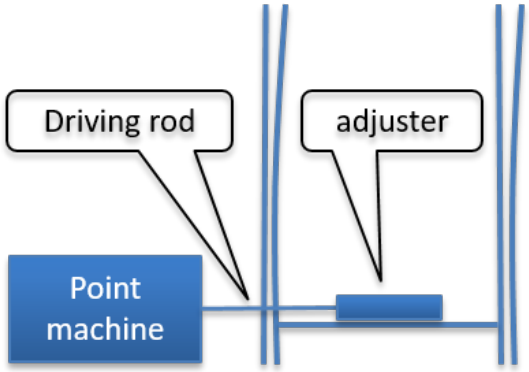

Figure 4. Rail-switch schematics

machine is composed of two sub-components: servo-motor and gear mechanism. The electrical motor acts as a power source. The gear mechanism is responsible for scaling down the angular velocity generated by the servo-motor, amplifying the torque generated by the servo-motor and transforming the rotational motion into a translational motion. The rail load is composed of two main components: the adjuster and

the tongue rails. The adjuster transfers the force generated by the motor (through the driving rod) to the rails. The adjuster connects the driving rod connected to the point machine to the rails. There is a delay between the time instances the driving rod and the adjuster start moving. This delay is controlled by two bolts on the driving rod. Tighter bolt settings means a smaller delay, while looser bolt settings produce a larger delay. The adjuster is connected to two rails that are moved from left to right or right to left, depending on the traffic needs. The motion of the rail is eased by a set of bearings and affected by the length of the rail and elasticity of the rail. Based on the technical specifications of the servo-motor and adjuster, we built Modelica models for them. Building a first-principle model for the rail proved to be challenging and hence we chose to learn a model for it from the measurement data.

The first step in learning a model for the rail is choosing a representation that is compatible with the rest of the model: it must have an interface (port or connector) compatible with the mechanical domain. The interface is characterized by two variables: a flow variable (force) and a non-flow variable (velocity). The product between the flow and non-flow variables has the interpretation of instantaneous power. Next we choose a set of constitutive equations that constraint the interface variables. We chose to represent the map involving the interface variable as a NN. Since such a map has a input and output, the next steps is determining which is which. Following the step described in section 2.2, we use the BLT representation to determine the input and the output of the NN. Note that any map (even a linear one) is sufficient to perform the BLT transform. It should not come as a surprise that the BLT transform indicates that the force is an output. Hence, we model the rail behavior by using a causal map $F = g(u; w)$, where $g : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a map described by a NN with one hidden layer:

$$g(u) = W^{[1]} \left(\tanh \left(W^{[0]} u + b^{[0]} \right) \right) + b^{[1]}, \qquad (26)$$

where, the input $u = [x, \dot{x}, \ddot{x}]$ is a vector containing the position, speed and acceleration, the output F is the force, and $w = \{W^{[0]}, b^{[0]}, W^{[1]}, b^{[1]}\}$ is the set of parameters of the map g.

4.2. Fault modes

We consuder four fault operating modes: left and right misaligned adjuster bolts, obstacle and missing bearings. These fault modes were reported to be of interest by a rail system operator we collaborated with. Obviously there are many other fault modes of interest at the level of the point machine for example. Such faults are more readily detected due to the rich instrumentation present at the servo-motor.

Misaligned adjuster bolts: In this fault mode the bolts of the adjuster deviate from their nominal position. As a result, the instant at which the drive rod meets the adjuster (and therefore the instant at which the switch rail starts moving) happens either earlier or later. For example in a left-to-right motion, if the left bolt deviates to the right, the contact happens earlier. The reason is that since the distance between the two bolts decreases, the left bolt reaches the adjuster faster. As a result, when the drive rod reaches its final position, there may be a gap between the right switch blade and the right stock rail. In contrast, if the left bolt deviates to the left the contact happens later. The model of the adjuster includes parameters that can set the positions of the bolts, and therefore the effects of this fault mode can be modeled without difficulty. Figures 5 and 6 show a comparison between the nominal behavior and the misaligned left and right bolts, respectively on the motor current and angular velocity.

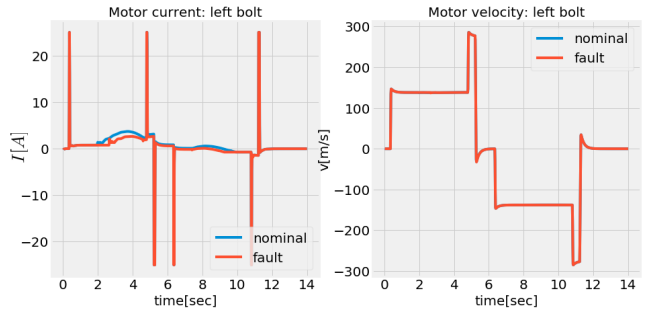

Figure 5. Effects of a misaligned left adjuster bolt on the motor current and angular velocity

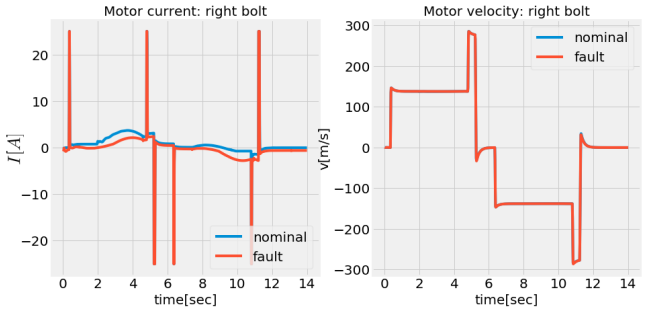

Figure 6. Effects of a misaligned right adjuster bolt on the motor current and angular velocity

Missing bearings: To minimize friction, the rails are supported by a set of rolling bearings. When they become stuck or lost, the energy losses due to friction increase. A component connected to the rail was included to account for friction. This component has a parameter that sets the value for the friction coefficient. By increasing the value of this parameter, the effect of the missing bearings fault can be simulated. Figure 7 shows a comparison between the nominal behavior and the missing bearing behavior on the motor current and angular velocity.

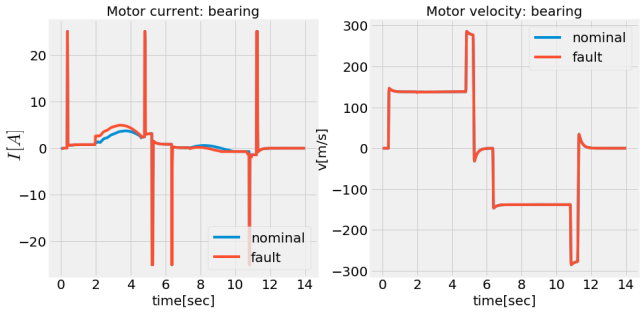

Figure 7. Effects of missing bearings on the motor current and angular velocity

Obstacle: In this fault mode, an obstacle obstructs the motion of the switch blades. In case the obstacle is insurmountable, a gap between the switch blades and the stock rail appears. The effect on the motor torque is a sudden increase in value, as the motor tries to overcome the obstacle. To model this fault we included a component that induces a localized, additional friction phenomenon for the switch blades. This component has two parameters: the severity of the fault and the position. For very high severity the switch blades cannot move beyond a certain position. Figure 8 shows a comparison between the nominal behavior and the obstacle present behavior on the motor current and angular velocity.

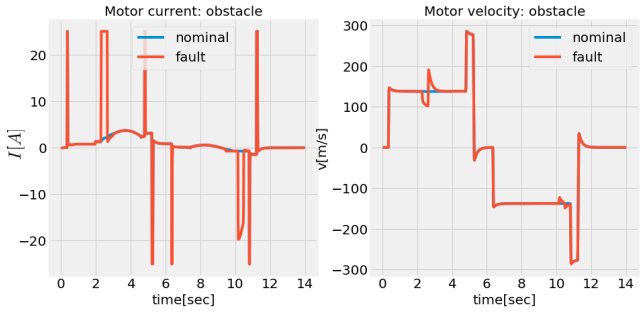

Figure 8. Effects of missing bearings on the motor current and angular velocity

4.3. Fault-diagnosis: partially known behavior

First we train the parameters of the rail model. Since the rail model is not directly impacted by the fault modes, we learn one single model that is valid for all modes. We chose the hidden layer dimension to be 20 for the NN modeling the rail. Hence we have a total of 100 parameters. To our knowledge, currently no deep learning platform supports DAE in the loop for the training process. The DAETools (Nikolic, 2016b) Python package does support DAE as dynamical models and enables gradient computations through sensitivity analysis. This requires though transforming the Modelica model into a form compatible with the DAETools formalism which is not a trivial process. Hence, we opted to use a gradient-free algorithm, and used a functional mockup unit (FMU) (Blochwitz et al., 2011) representation of the rail-switch model that was imported in Python and integrated in an least-square optimization process. In particular, we use Powell algorithm, which is the closest gradient-free optimization algorithm to gradient-based one. The training data corresponds to the nominal rail behavior, and consist in motor current, angle and angular velocity measurements. The inputs to the server motor are pre-designed reference signals that ensure a specific angular velocity profile for the rail. A 7 sec reference signal profile ensures the motion from left to right of the rail. A reversed reference profile, ensures the rail motion from right to left. The output measurements are time series over 14 sec, sampled at 0.05 sec time period.

Since the fault scenario does not directly affect the rail, only nominal data is used to train the rail model parameters. Using the Powell algorithm, we solved the following optimization problem:

$$\min_{W^{[1]},b^{[1]},W^{[0]},b^{[0]}} \quad \frac{1}{N}\sum_{i=1}^{N} \|y(t_i) - \hat{y}(t_i)\|^2 \tag{27}$$

$$\text{subject to:} \quad \boldsymbol{F}(\dot{z}(t_i), z(t_i)) = 0 \tag{28}$$

$$y(t_i) = \boldsymbol{h}(z(t_i) \tag{29}$$

where $\boldsymbol{F}(\dot{z}, z) = 0$ is the DAE corresponding to the rail switch model that includes the rail representation shown in Eq. (26), and $\boldsymbol{h}(z)$ is the measurement model that selects the motor current, angle and angular velocity from the model variables. The variables $y(t_i)$ and $\hat{y}(t_i)$ are measured and simulated output measurements, respectively. The variances of the output prediction errors were estimated to be: 0.05, 0.74, and 0.57 for the motor current, angle and velocity, respectively.

Next we train the parameters of the classifier as described in Algorithm 1. For each of the fault modes we generated 1000 time series as training data. The fault data was generated by selecting some fault parameters and adding noise to the outputs. In particular, for the left bolt fault mode we set a deviation from its nominal value of 50 mm, for the right bolt fault mode we set a 200 mm deviation from its nominal value, for the bearing fault mode we the viscous coefficient at 5000 Ns/m, and we set an obstacle at 10 cm from the initial rail position, with a viscous coefficient equal to 10^5 Ns/m affecting the rail motion. The noise free faulty behavior corresponding to the four fault modes are shown in Figures 5-8. The noise added to the outputs was chosen as zero mean Gaussian noise with variances determined by the trained model, as shown above.

We split the data into training (60%) and test (40%) data. The probabilities $q(\theta = j|y^{(i)}_{0:T})$ follow from the time series labels: $q(\theta = j|y^{(i)}_{0:T}) = 1$ if the time series $y^{(i)}_{0:T}$ corresponds to

mode j, and zero otherwise. To define the loss function, we calculated the probabilities $p(y^{(i)}_{0:T}|\theta = j)$ by approximating them using model simulations. Namely, for each model j, we simulated the rail-switch model by activating the j^{th} fault and generating the output time series $\hat{y}^{(j)}_{0:T}$. It follows that

$$p(y^{(i)}_{0:T}|\theta = j) = \frac{1}{\sqrt{(2\pi)^3|\Sigma|}} e^{-\frac{1}{2}(y^{(i)}_{0:T}-\hat{y}^{(j)}_{0:T})^T \Sigma^{-1}(y^{(i)}_{0:T}-\hat{y}^{(j)}_{0:T})}, \quad (30)$$

where Σ is a diagonal matrix with diagonal entries determined by the output noise variances, and $|\Sigma|$ being the determinant of Σ. Let $q_{ij} = q(\theta = j|y^{(i)}_{0:T})$ and $p_{ij} = p(\theta = j|y^{(i)}_{0:T})$. The final step is the compute the switch parameters η_j that minimizes the cross-entropy loss function:

$$\min_{\eta_j} \frac{1}{N} \sum_{i,j}^{N,5} q_{ij} \log p_{ij}, \quad (31)$$

where $p_{ij} = \eta_j p(y^{(i)}_{0:T}|\theta = j)/(\sum_{l=1}^{5} \eta_l p(y^{(i)}_{0:T}|\theta = l))$, and N = 3000. We used Autograd to compute the gradient of the loss function, and Adams algorithm to compute the optimal solution described in Table 1. The confusion matrices for the training and testing data are shown in Figures 9 and 10.

Parameters	Values
η_1	0.00955831
η_2	0.00994298
η_3	0.34316536
η_4	0.58860535
η_5	0.06784462

Table 1. Optimal solution for the hybrid diagnoser

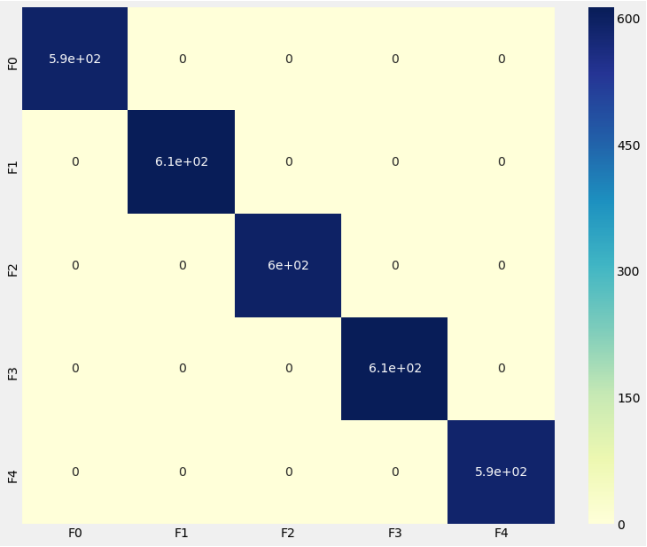

Figure 9. Confusion matrix: training data

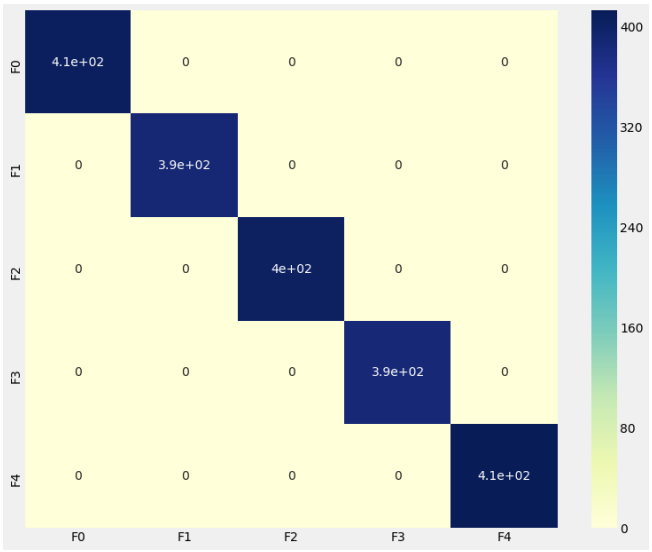

Figure 10. Confusion matrix: testing data

4.4. Fault diagnosis: neural network based classifier

We trained a NN-based classifier using the same data set. In general, it is difficult to find the best and the most parsimonious NN architecture that generates good results. We used a trial and error process to converge to a NN architecture that gives accurate results. Using the 14 sec time series as input samples proved to be a bad idea. The 5000 sample were not enough for the ten of thousands of parameters of the NN. We recall that the number of columns of the first layer of the NN is given by the input size. Hence, we had to reduce the number of inputs. Instead of using an autoencoder which is typically greedy for data, we trained a random forest classifier and used its feature importance output to select 27 entries of the time series that contain relevant information for differentiating between the fault modes. We again employed a trial and error process to converge to the minimal number of features and a parsimonious NN architecture that is able to learn an accurate classifier. We ended up with a NN with one hidden layer of size 15 and with an output layer of size 5 that uses a softmax function as an activation function. Hence we have a total number of 500 training parameters. Although we cannot guarantee that there is no simpler NN architecture, empirically we have noticed that the prediction accuracy decreases for hidden layer sizes smaller than 15. After training the NN parameter we ended up with a classifier that has similar accuracy performance as the one shown in the previous section.

4.5. Discussion

When including the partial model, the complexity of the classification is transferred from learning a potentially complex classifier to training a regression model for the missing com-

ponent. Hence the potential to reduce complexity. The classification problem for the partial model knowledge case is much simpler and hence more easily to train. In addition, we escape the feature selection step that is typically an ad-hoc process. In addition, since we maintain the physical interpretation of the model (at least in part), there are opportunities to further investigate the consequences of faults to other system components as faults progress. That is, we can use the model for prognostics. Machine learning algorithms for prognostics are hungry for data; data that in many cases is not available. The partial model has a regularization effect on the learning algorithm, and hence it is an avenue for dealing with small data sets and limiting this way the overfitting. The classification results for both the hybrid and machine learning architecture were perfect. This is most likely due to the use of simulated data for the faults modes. Still, we expect our approach to give reasonable results on experimental data as well, using a complexity reduced classifier.

5. CONCLUSIONS

In this paper we discussed the classification problem based on data generated by a partially known physical system. Unlike standard classification problems, where the classifier ignores any knowledge about the physical system, our goal was to integrated this information in the classifier design. We demonstrated that the classification problems can be converted into a set of regression problems and a set of dimensionally reduced classification sub-problems. We introduced two algorithms for learning a classifier, each one corresponding to an assumption on the measurement noise. We showcased our approach in the context of fault diagnosis for a rail switch system.

ACKNOWLEDGMENT

This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) Award HR00111890037 Physics of AI (PAI) Program.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Arulampalam, M. S., Maskell, S., & Gordon, N. (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, *50*, 174–188.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Claub, C., Elmqvist, H., ... Augustin, S. (2011). The functional mockup interface for tool independent exchange of simulation models. In *In proceedings of the 8th international modelica conference.*

de Kleer, J., Mackworth, A., & Reiter, R. (1992). Characterizing diagnoses and systems. *"Journal of Artificial Inteligence"*, *56*(2–3), 197–222.

Gertler, J. (1998). *Fault-detection and diagnosis in engineering systems*. New York: Marcel Dekker.

Isermann, R. (2005). Model-based fault-detection and diagnosis - status and applications. *Annual Reviews in Control*, *29*(1), 71 - 85.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, *82*(Series D), 35–45.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*. Retrieved from `http://arxiv.org/abs/1412.6980`

Maclaurin, D., Duvenaud, D., Johnson, M., & Adams, R. P. (2015). *Autograd: Reverse-mode differentiation of native Python.* Retrieved from `http://github.com/HIPS/autograd`

Matei, I., de Kleer, J., & Minhas, R. (2018, June). Learning constitutive equations of physical components with constraints discovery. In *Proceedings of the ieee 2018 american control conference (acc 2018).*

Matei, I., Ganguli, A., Honda, T., & de Kleer, J. (2015, Aug). The case for a hybrid approach to diagnosis: A railway switch. In *Proceedings of the 26th international workshop on principles of diagnosis (dx-2015)* (pp. 225–232).

Nikolic, D. D. (2016a, April). Dae tools: equation-based object-oriented modelling, simulation and optimisation software. *PeerJ Computer Science*, *2*, e54. Retrieved from `https://doi.org/10.7717/peerj-cs.54` doi: 10.7717/peerj-cs.54

Nikolic, D. D. (2016b, April). Dae tools: equation-based object-oriented modelling, simulation and optimisation software. *PeerJ Computer Science*, *2*, e54.

Patton, R. J., Frank, P. M., & Clark, R. N. (2000). *Issues of fault diagnosis for dynamic systems*. Springer-Verlag London.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, *abs/1609.04747*. Retrieved from `http://arxiv.org/abs/1609.04747`

Subramanian, V. (2018). *Deep learning with pytorch: A practical approach to building neural network models using pytorch* (1st ed.). Packt Publishing.