Classification-based Diagnosis Using Synthetic Data from Uncertain Models

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

[1,2,3,4] *Palo Alto Research Center, Palo Alto, CA*
ion.matei@parc.com
maksym.zhenirovskyy@parc.com
dekleer@parc.com
alexander.feldman@parc.com

Abstract

Machine learning based diagnosis engines require large data sets for training. When experimental data is insufficient, system models can be used to supplement the data. Such models are typically simplified and imprecise, hence with some degree of uncertainty. In this paper we show how to deal with uncertainty in synthetic training data. The data is produced using a model with uncertainties. The uncertainties originate from inaccurate parameter values or parameters that take different values based on the mode of operation. We demonstrate how techniques from the uncertainty quantification field can be used to reduce the numerical complexity of the training algorithm. In particular, we use generalize polynomial chaos to efficiently approximate the loss function. In addition, we present a neural network architecture specifically designed to deal with uncertainties in the training data. As an illustrative example, we show how our approach can be used to detect faults in an elevator system.

1. Introduction

Our goal is to train a machine-learning based classifier to diagnose faults in a physical system. In our scenario, we do not have sufficient training data, but we have a model of the system. The model suffers from uncertainties whose sources are traced to imprecise parameter values or modeling simplifications. This scenario is rather common when dealing with physical systems. Many such systems become faulty rarely and hence often little data describing faulty behavior is available. In addition, they can be very expensive, and their operators would like to detect faults early on, to prevent catastrophic failures. We encountered such cases in some of our previous work (Matei, Ganguli, Honda, & de Kleer, 2015). There is an additional scenario under which parameters are uncertain. Consider an elevator car. Its mass varies according to the number of passengers inside the car. If the car mass is not directly measurable, rather than having a fixed (average) mass, we can model it as a random vector with some probability distribution. The reader may ask why do we have to use a machine learning classifier when we could use a model-based diagnosis method since a model in available. There is a trade-off between the two approaches. Classifiers are agnostic to the type of physical system and all the training effort is done off-line. They do require large data sets though. In turn, model-based methods are more sensitive to the type of system. For example, for linear systems with Gaussian noise diagnosis engines based on Kalman filters (Kalman, 1960) work well. For nonlinear systems extensions of the Kalman filter (extended or unscented) can be used. They do not always work well. Extended Kalman filter requires the Jacobian of the system model. This is rather difficult to compute for a complex system. The unscented Kalman filter is rather sensitive to its hyper-parameters. Alternatively, we can use use a particle filter (Arulampalam, Maskell, & Gordon, 2002) that deal well with nonlinear systems and non-Gaussian noise. The computational effort required to implement such filter online may be prohibitive though.

For the faults for which training data is insufficient or missing, we augment the model with physics-based failure behavior corresponding to the respective faults. The augmented model will have the capability to generate data that reflects the behavior of the system under these faults. To train the classifier, we first need to execute a data augmentation process. The process will generate training data that include the system behavior under the set of faults of interest. We use model simulations to achieve this objective. Next, we need to use a training algorithm that addresses the uncertainty in the training data that originates in model. The training data will be affected by uncertainty since it was generated using an uncertain model. Hence the training algorithm must take this into account. We draw inspiration from the field of uncertainty quantification (UQ) (Smith, 2013), and use a method based on generalized chaos polynomial (gPC) expansions (O'Hagan, 2013; Xiu, Lucor, Su, & Em Karniadakis, 2003). This method reduces the numerical complexity of the training algorithm. The gPC expansions enable efficient evaluation of the training cost function using quadrature-based approximations. A brute force approach to deal with uncertainties is based on the Monte-Carlo method. It relies on a potentially large number of repeated random sampling to obtain numerical results. Hence, it is computationally expensive. *In this paper, reduced numerical complexity translates to reduced number of model simulations.*

Paper structure: We start with a description of the problem in Section 2. We continue with introducing concepts related to the gPC framework in Section 3. In Section 4 we show the structure of the training algorithm that deals with data uncertainty. In the same section we introduce a neural network (NN) architecture adapted to our problem setup. Section 5 demonstrates our approach when diagnosing faults for an elevator with random car mass.

2. Problem setup

We consider physical systems whose behavior can be described by a set of ordinary differential equations (ODEs) of the form

$$\dot{s} = f(s,u;\theta), \quad (1)$$
$$z = h(s,u;\theta), \quad (2)$$

where s represents the state, u is a vector of inputs, z is a vector of outputs, and θ denotes the vector of parameters of the system. The vector θ is a vector valued random variable. There are several sources of uncertainty in the model. For example, one source of uncertainty has its origin in the manufacturing processes. They are almost never deterministic. Hence, if we have the parameter values of the model components, they are not deterministic. Another source of uncertainty comes from the modeling process itself. We make simplifications or learn representations for components when technical specifications are incomplete or missing. We showed in (Matei, de Kleer, & Minhas, 2018) how we can learn feasible acausal models that are not based on first principles. These models are rarely perfect. We assume that the probability distribution of θ is known. We make an additional assumption. Namely, that the model was augmented such that the behavior corresponding to a set of faults $\mathcal{F} = \{f_0, f_1, \ldots, f_L\}$ can be simulated. Fault f_0 denotes the nominal behavior. The physics-based fault augmentation process adds additional equations to the model. These new equations are dependent on parameters whose activation induces the simulated faulty behavior. The type of faults introduced are domain dependent. We cover electrical (short, open connections, parameter drifts), mechanical (broken flanges, stuck flanges, torque losses due to added friction, efficiency losses), or fluid (blocked pipes, leaking pipes) domains. Fault augmented models enabled us to execute a number of system analytics tasks, ranging from fault diagnosis (Minhas et al., 2014), reliability analysis (Honda et al., 2014) to maintenance scheduling (Saha et al., 2014).

We use the model to generate data that correspond to each of the fault modes. We can use a sliding window approach to extract time series $\{s_{t_k:t_k+T}, z_{t_k:t_k+T}\}$ that can be regarded as raw data, where t_k denotes time samples, and T represents the window size (Figure 1). The raw data is processed further to extract features meaningful for the classification process. Hence we obtain a training data set $\mathcal{D} = \{(x^{\{i\}}, y^{\{i\}})\}$, where $x^{\{i\}}$ corresponds to a time series $\{s_{t_k:t_k+T}, z_{t_k:t_k+T}\}$, $y^{\{i\}}$ encodes a particular fault mode, and i denotes a training example. In particular, $y^{\{i\}}$ is a $L+1$ dimensional binary vector, with $y_j^{\{i\}} = 1$ when $x^{(i)}$ was produced under fault mode f_j, and zero otherwise. The data-set $\mathcal{D}$ depends on the system parameters θ and hence it inherits the uncertainty in θ: $x^{\{i\}} = x^{\{i\}}(\theta)$. Data set $\mathcal{D}$ can include experimental data as well, but that specific chunk of data will not be dependent on θ in training algorithm described in the following section.

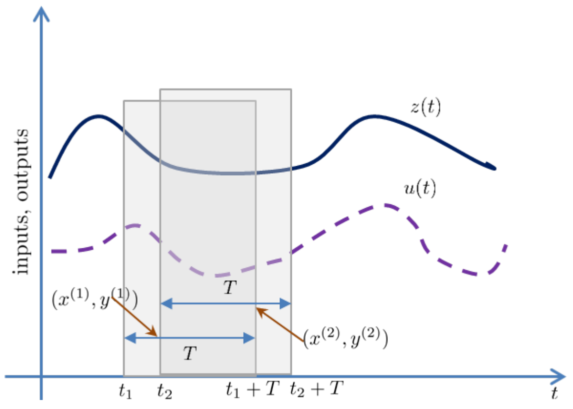


Figure 1. Training data: a time window is moved over the inputs and outputs of the system; each window generates a feature vector

The purpose of the training algorithm is to learn the classifier model formulated as a mapping $y = g(x;\beta)$ such that $g(x^{\{i\}};\beta)$ and $y^{\{i\}}$ are close to each other with respect to some metric. The classifier parameter vector β is computed by minimizing a sum of loss functions $L(\beta;x,y)$ with respect to β. The loss function measures the discrepancy between the classifier's prediction and the true output y. A typical loss function for classification is the cross entropy: $L(\beta;x,y) = -\sum_{j=1}^{L+1} y_j \log \hat{y}_j$, where $\hat{y} = g(x;\beta)$. The parameter vector β is derived by solving the optimization problem

$$\hat{\beta} = \arg\min_{\beta} \mathcal{L}(\beta;\mathcal{D}), \quad (3)$$

where $\mathcal{L}(\beta;\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|} L(\beta; x^{\{i\}}, y^{\{i\}})$. In our setup, since the training data depends on θ and hence is stochastic, the cost function is stochastic as well. Hence, the optimization problem is reformulated by averaging over the values of θ

$$\hat{\beta} = \arg\min_{\beta} \mathbb{E}_{\theta}\left[\mathcal{L}(\beta;\mathcal{D}(\theta))\right], \tag{4}$$

where the expectation is taken with respect to the distribution of θ. The key part in solving Eq. (4) is the evaluation of the expectation $\mathbb{E}_{\theta}\left[\mathcal{L}(\beta;\mathcal{D}(\theta))\right] = \int \mathcal{L}(\beta;\mathcal{D}(\theta))dP_{\theta}$, where dP_{θ} is the probability measure of θ. A brute-force approach based on Monte-Carlo is computationally expensive since it requires a large number of simulations. This approach will not scale with the size of the system and the number of parameters. Our goal is to provide an efficient way to approximate the expectation shown in (4).

3. Preliminaries - Generalized Polynomial Chaos Expansions

To train a classifier we need to solve the optimization problem (4). This in turn requires finding an efficient way to evaluate the expectation in terms of the probability distribution of θ. To achieve this we make use of concepts from the theory of uncertainty quantification. In particular, we use the gPC framework, briefly introduced in what follows.

Based on the homogeneous chaos theory (Wiener, 1938) and subsequent generalizations using Wiener-Askey scheme (Ogura, 1972; R. H. Cameron, 1947), a second-order (finite variance) random process $Z(\omega)$ can be represented as the infinite sum

$$Z = \sum_{i=0}^{\infty} Z_i \psi_i(\xi), \tag{5}$$

where ξ is a random variable and $\psi_i(\xi)\}$ are orthogonal polynomials satisfying the orthogonality relation

$$\langle \psi_i, \psi_j \rangle = ||\psi_i||^2 \delta_{ij}, \tag{6}$$

where δ_{ij} is the Kronecker delta, and $\langle\cdot,\cdot\rangle$ denotes the inner product

$$\langle f(\xi), g(\xi)\rangle = \int f(\xi)g(\xi)W(\xi)d\xi, \tag{7}$$

with $W(\xi)$ the weighting function. Interestingly, some types of orthogonal polynomials from the Askey-scheme have weighting functions that are the same as the probability functions of certain types of random variables (Table 1). If Z is m-dimensional, we associate an independent random variable ξ_i to each entry of Z. In this case $Z = \sum_{i=0}^{\infty} Z_i \boldsymbol{\psi}_i(\xi)$, where $\xi = [\xi_1, \ldots, \xi_m]$. A straightforward way to obtain the basis $\{\boldsymbol{\psi}_i(\xi)\}$ is to construct tensor products of one-dimensional polynomials corresponding to each random variable ξ_i, namely $\boldsymbol{\psi}_i(\xi) = \psi_{j_1^i}(\xi_1)\ldots\psi_{j_m^i}(\xi_m)$. For practical purposes, the infinite chaos expansion is truncated to a finite sum. If P is the highest order of the scalar polynomial ψ, the total number of expansion terms is $M+1$, with $M = (m+P)!/(m!P!) - 1$ (in the one-dimensional case $m = 1$, we have that $M = P$). A significant increase in P and m will have a high impact on the computational effort necessary for evaluating inner products of the form (7). However, since the convergence of the Chaos expansion can be exponential (R. H. Cameron, 1947), we can choose expansions of reasonable dimensions.

Table 1. Correspondence between the type of Wiener-Askey polynomial chaos and its underlying variable.

ξ	$\psi(\xi)$	Support
Gaussian	Hermite-Chaos	$(-\infty, \infty)$
Gamma	Laguerre-Chaos	$[0, \infty)$
Beta	Jacobi-Chaos	$[a, b]$
Uniform	Legendre-Chaos	$[a, b]$
Poisson	Charlier-Chaos	$\{0, 1, 2, \ldots, \}$
Binomial	Krawtchouk-Chaos	$\{0, 1, 2, \ldots, N\}$
Negative Binomial	Meixner-Chaos	$\{0, 1, 2, \ldots, \}$
Hypergeometric	Hahn-Chaos	$\{0, 1, 2, \ldots, N\}$

It is well known that at the core of Gauss-quadrature schemes for numerical integration are orthogonal polynomials. In our context, this provide an efficient way to approximate expectations with respect to the probability distribution of ξ. For example, let ξ be a scalar standard Gaussian random variable, and let $\psi_i(\xi)$ be the associate orthogonal polynomials. Then the expectation of any function $h(\xi)$ can be approximated using the Gauss quadrature:

$$\mathbb{E}[h(\xi)] \approx \sum_{n=1}^{N} w_n h(\xi^{(n)}), \tag{8}$$

where $\xi^{(n)}$ are collocation points which are the roots of the N-degree polynomial ψ_N. As an example, if $\xi \sim \mathcal{N}(0,1)$ and therefore $\{\psi_i(\xi)\}$ are (probabilists') Hermite polynomials, we can apply the Gauss-Hermite quadrature and obtain

$$\mathbb{E}[h(\xi)] \approx \sum_{n=1}^{N} w_n h(\xi^{(n)}\sqrt{2}), \tag{9}$$

where $\xi^{(n)}$ are collocation points, which are the roots of the N-degree (physicists') Hermite polynomial $\tilde{\psi}_N(\xi) = 2^{\frac{N}{2}}\psi_N(\xi\sqrt{2})$, with associated weights $w_n = \frac{2^{N-1}N!}{N^2[\tilde{\psi}_{N-1}(\xi^{(n)})]^2}$. We are not bound to a particular choice of chaos orthogonal polynomials. In the case $\xi \sim \mathcal{U}(-1,1)$ with $\{\psi_i\}$ Legendre polynomials, we can apply the Gauss-Lengendre quadrature to evaluate the expectation. As an example, the estimation error for the mean of a uniform random variable in the interval [-1,1] using Gauss-Lengendre quadrature and 10 quadrature points is 1.67e-15. By comparison, by randomly sampling 10 points (e.g., Monte Carlo approach), the average estimation error is 0.146.

For higher dimensions $\xi = (\xi_1, \ldots, \xi_m)$, an immediate solution is to create a grid based on the Cartesian product

$\times_{j=1}^{m}\{\xi_j^{(n)}\}_{n=1}^{N}$, where $\{\xi_j^{(n)}\}_{n=1}^{N}$ are collocation points corresponding to the one dimensional case. As m and P increase, the size of the grid increases as well. To deal with the exponential increase in the number of tensor product terms (P^m), sparse grid quadrature methods can be applied (Holtz, 2008). These methods are based on using certain combinations of tensor products of one-dimensional quadrature rules. They can exploit the smoothness of $\{\psi_i\}$ to overcome the curse of dimensionality to certain extent.

4. Training algorithm

Feature vectors are generated through numerical simulations of the physical system model. They are functions of the vector valued random variable θ, that is, $x = x(\theta)$. To use the gPC expansion, we first need to represent θ in terms of ξ. Assuming for simplicity θ is a scalar, the gPC expansion of θ is given by $\theta = \sum_{i=0}^{M}\theta_j\psi_j(\xi)$. The coefficients θ_j follow from the orthogonality property, provided a set of expectations in terms of θ and ξ can be evaluated. However, since θ and ξ may actually belong to different probability spaces, with different event spaces and σ-algebras, we first need to map them on the same probability space by applying a measure transformation. What this means is that we need to represent θ and ξ as a function of a new random variable. Let $dF_\theta(\theta)$ and $dF_\xi(\xi)$ be the probability measures of θ and ξ, respectively. We can define the random variable $U \sim \mathcal{U}(0,1)$ and impose $du = dF_\theta(\theta) = dF_\xi(\xi)$. Recall that du is in fact the pdf of U, since $dF_U = f(u)du = du$. Moreover, this gives us the transformations $\theta = F_\theta^{-1}(u)$ and $\xi = F_\xi^{-1}(u)$. Thus, the coefficients of the expansion $\theta = \sum_{i=0}^{M}\theta_i\psi_i(\xi)$ can be computed as

$$\theta_j = \frac{1}{\|\psi_i^2\|}\langle\theta,\psi_j(\xi)\rangle = \frac{1}{\|\psi_j^2\|}\int\theta\psi_j(\xi)dF_\xi(\xi) =$$

$$\frac{1}{\|\psi_j^2\|}\int_0^1 F_\theta^{-1}(u)\psi_j\left(F_\xi^{-1}(u)\right)du. \tag{10}$$

Due to the choice of distribution for U, the above integral can be accurately approximated using Gauss-Legendre quadrature. Using this procedure, we can express the feature vectors as a function of ξ, that is, $x = x(\xi)$. Representing θ in terms of ξ enable the approximation of the expectation using quadratures. In the case θ is a vector, ξ is a vector as well, where its entries are independent random variables. It means that their joint pdf can be evaluated as a product of pdfs corresponding to scalar random variables. This simplifies the evaluation of the expectation induced by the vector valued random variable ξ. The expectation in the optimization problem (4) can be approximated as

$$\mathbb{E}_\theta\left[\mathcal{L}(\beta;\mathcal{D}(\theta(\xi)))\right] =$$

$$\mathbb{E}_\xi\left[\mathcal{L}(\beta;\mathcal{D}(\xi))\right] \approx \sum_{n=1}^{N} w_n\mathcal{L}(\beta;\mathcal{D}(\xi^{(n)})), \tag{11}$$

where $\xi^{(n)}$ are collocation points and w_n are coefficients that depend on the type of quadrature used. Table 1 enumerates what quadrature are used based on the probability distribution of ξ. Therefore, the optimization problem (4) can be approximated as

$$\min_\beta \sum_{n=1}^{N} w_n\mathcal{L}(\beta;\mathcal{D}(\xi^{(n)})). \tag{12}$$

This implies that for each collocation point $\xi^{(n)}$ we need to simulate the model of the physical system to generate a set of features. We note a tradeoff between the accuracy of the approximation and the numerical effort incurred by the simulations. The use of gPC expansions guides the choice of collocations points, reducing the numerical effort as compared to a Monte Carlo approach. Typical training algorithms are based on stochastic gradient descent and its variants (Kingma & Ba, 2014; Dozat, 2013). An instance of the gradient descent algorithm for our optimization problem is given by

$$\beta_{k+1} = \beta_k - \alpha\sum_{n=1}^{N} w_n\nabla_\beta\mathcal{L}(\beta_k;\mathcal{D}(\xi^{(n)})), \tag{13}$$

where α is the (possible time varying) iteration step size. This algorithm requires the evaluation of the gradient of $\mathcal{L}$ at each collocation point $\xi^{(n)}$. If we choose to model the classifier as a neural network, the current platforms for training large scale classifiers, such as Keras, Tensorflow or Pytorch can be used, provided some custom made layers are built. To be more concrete, let us first revisit the cost function:

$$\mathbb{E}\left[L(\beta;x(\xi),y)\right] = -\sum_{j=1}^{L+1}\mathbb{E}\left[y_j\log\bar{y}_j(\xi)\right] \approx$$

$$-\sum_{j=1}^{L+1}\sum_{n=1}^{N} w_n y_j\log\bar{y}_j(\xi^{(n)}), \tag{14}$$

where $\bar{y}(\xi) = g(\beta;x(\xi))$. We further have

$$\mathbb{E}\left[L(\beta;x(\xi),y)\right] \approx$$

$$-\sum_{j=1}^{L+1} y_j\log\left(\prod_{n=1}^{N}\bar{y}_j(\xi^{(n)})^{w_n}\right) = -\sum_{j=1}^{L+1} y_j\log\hat{y}_j, \tag{15}$$

where $\hat{y}_j = \prod_{n=1}^{N}\bar{y}_j(\xi^{(n)})^{w_n}$. This formula guides us to a particular type of neural network architecture as shown in Figure 2. We start with a neural network that has as input a feature vector and has as output a softmax function that generates a binary vector of dimension $L+1$. We make N copies of this neural network, each of then receiving as input $x^{(n)} = x(\xi^{(n)})$. All copies share the same parameters β and generated outputs $y^{(n)[1]}$ [1]. Next, each output $y^{(n)[1]}$ passes through a nonlinear layer that raises its values to the w_n power, that is,

[1]Notation $y^{(n)[1]}$ refers to the n^{th} subvector of $y^{[1]}$, where the notation $[\cdot]$ represents the layer index. By $y_i^{(n)[1]}$ we represent the i^{th} entry of $y^{(n)[1]}$

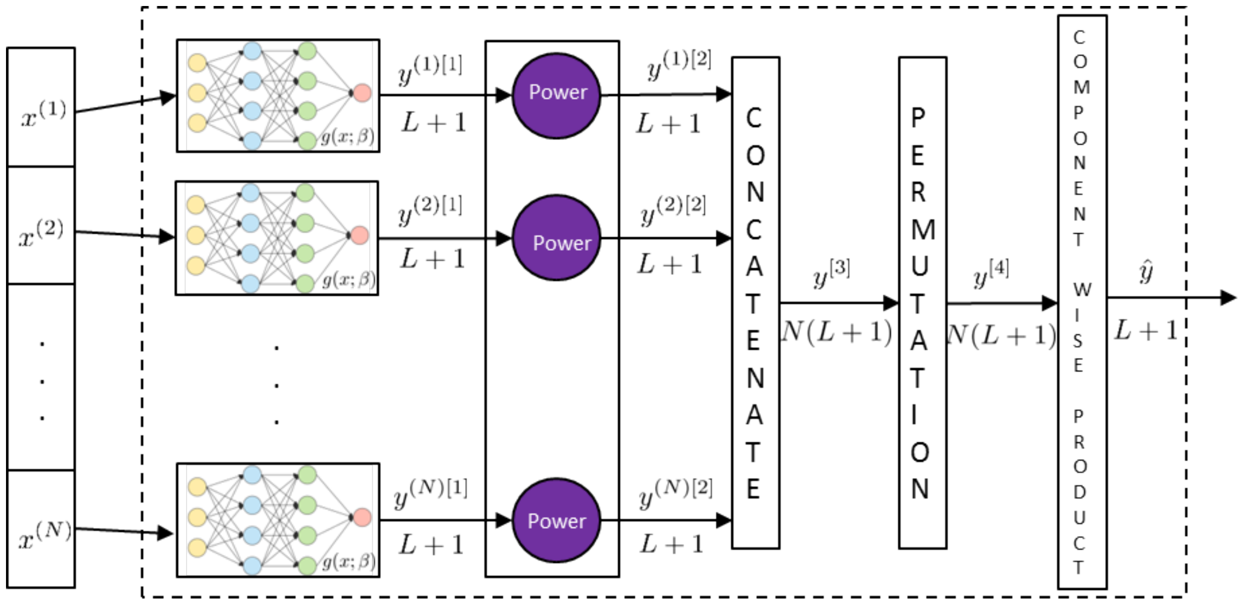


Figure 2. Neural network architecture when using training data with uncertainties

$y^{(n)[2]} = \left(y^{(n)[1]}\right)^{w_n}$. These outputs are concatenated into a large vector $y^{[3]} = [y^{(1)[2]}, \dots, y^{(n)[2]}]$, which is passed through a linear layer with constant weights matrix, $y^{[4]} = Py^{[3]}$, where P is a permutation matrix. Matrix P is defined in such a way that j^{th}, $L+1$ dimensional component of $y^{[4]}$ has the form $y^{(j)[4]} = [y_j^{(1)[2]}, \dots, y_j^{(n)[2]}]$. The last layer computes products of the elements of each N dimensional sub-vectors of $y^{[4]}$, that is, $\hat{y}_j = \prod_{n=1}^{N} y_n^{(j)[4]}$, where j denotes the j^{th} sub-vector of $y_n^{[4]}$. There are $L+1$ such sub-vectors.

Note that although we execute more computations, the number of parameters does not increase with the number of collocation points. This is because first layer's NNs share the same parameters β, and all other layers have known parameters. In addition, the operations performed by these last layers are differentiable and hence we can generate the necessary gradients for the backpropagation algorithm. Hence we can use main stream NN training platform such as Tensorflow, Keras or Pytorch for training.

The same idea can be applied to non-parametric statistical models, such as decision tree or random forests. The training algorithms for such models use loss functions whose expectation when dealing with uncertain data can be approximated using the gPC framework.

5. Illustrative example

We apply our approach for diagnosis faults in an elevator system. The block diagram of the system is shown in Figure 3. A velocity reference based on the car position is transmitted to the velocity controller. This ensures that the electric motor acting on the sheave follows the velocity reference. A typical velocity reference profile is shown in Figure 4. After a command is given, the car starts to accelerate until it reaches a cruising velocity. After passing a position sensor marking the approach to the destination, the car starts to decelerate to a complete stop. A model of the elevator was implemented

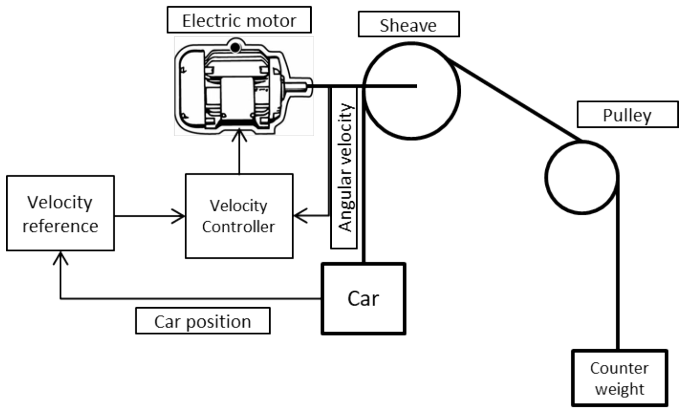


Figure 3. Block diagram of the elevator system

using Modelica language (Fritzson & Bunus, 2002). Our objective is to train a classifier able to detect two faults: motor wear and presence of an obstacle in the shaft that impedes the car's motion. The model was augmented to allow simulation of the two faults. The fault components were added on top of the original model. It is suitable to cases where access to the details of the model is not permitted. We used this approach in some of our previous fault augmentation work (Honda et al., 2014; Saha et al., 2014). The motor wear was

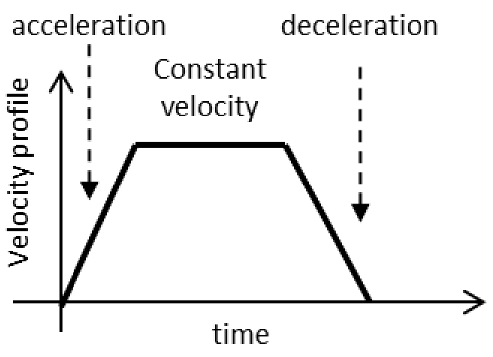


Figure 4. Typical velocity profile

Table 2. Class definitions for the system behavior.

Classes	Motor efficiency	Friction parameter
(1)-Nominal	[1,0.9]	[0, 0.1]
(2)-Fault 1	(0.9,0.5]	0
(3)-Fault 2	(0.5, 0]	0
(4)-Fault 3	1	(0.1, 0.5]
(5)-Fault 4	1	(0.5, 1]

implemented by modeling a torque efficiency loss. The obstacle wear was modeled by including a localized increase in friction. To overcome the friction, the motor has to generate more torque in order to track the velocity reference. The severity of the two fault modes is controlled by two parameters. A motor efficiency parameter determines the wear of the electric motor. A value of one for the parameter means that the motor functions at full efficiency, while a zero value describes a complete failure of the motor. A zero value for the friction is equivalent to the nominal case (no friction), while a value one generates a friction force that blocks completely the motion of the car. We define a total of five classes that reflect the behavior of the elevator system, as shown in Table 2. We considered single faults only, although we can easily extend the analysis to the double faults case, by executing additional simulations.

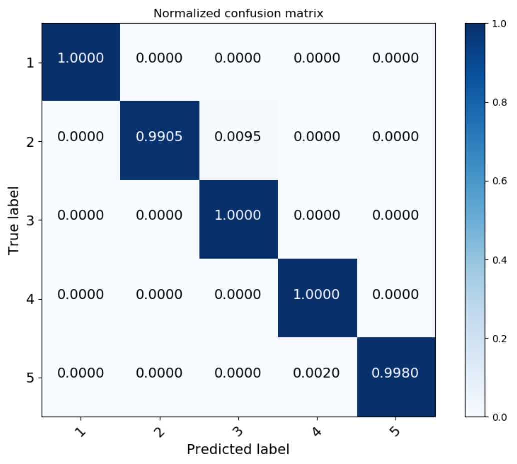
Next we generate simulated data that reflects the behavior of the system under the five operating modes. We assume we only have access to the empty car mass (100 Kg), and model the car mass during operation as random variable with a uniform distribution between 100 Kg and 300 Kg. This represents the uncertainty in our model. Rather then generating simulated data for a large number of mass values we apply the uncertainty quantification-based approach to generate a relatively small number of mass values. According to Table 1, we use chaos-Legendre polynomials to approximate the optimization cost function. We choose an expansion with ten terms (N=10). Hence we need to generate data for ten mass values only. The weights w_n of the expectation approximation are computed according to the formula

$$w_n = \frac{1}{(1-z_n^2)P'(z_n)^2},\ n = 1\ldots,10, \tag{16}$$

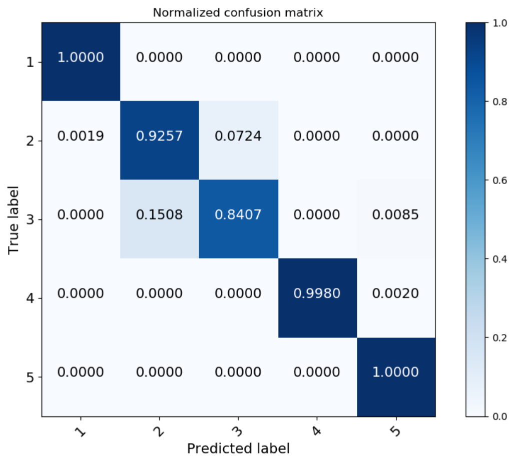
where z_n are the roots of the tenth order Legendre polynomial $P(z) = \frac{1}{256}(46189z^{10} - 109395z^8 + 90090z^6 - 30030z^4 + 3465z^2 - 63)$ and $P'(z)$ is the derivative of $P(z)$ with respect to z. The car mass values are given by $\frac{b-a}{2}z_n + \frac{a+b}{2}$, where z_n are the roots of $P(z)$, a=100, and b=300. For each mass value we executed a number of 2000 simulations covering the five classes. A simulation interval is given by the motion of the car from an initial position to a final position and the return. Each class has a number of 400 simulations obtained by varying the fault parameters in the intervals associated to each class. Hence we executed a number of 20,000 (10×5×400) simulations. As test data we simulated the model for ten mass values not included in the previous set of mass values. Using a similar approach as in the case of the training data, we generated a number of 100 simulations per each class and mass value. Hence the test data contains 5,000 (10×5×100) samples. The following variables are assumed measurable: car *position* and *velocity*, motor *current*, *control signal* fed to the electric motor. A training example contains typical feature extracted from time series representing the four variables over one simulation interval: `min`, `max`, `mean`, `standard deviation` and `median` value. We obtain this way a feature vector of size twenty (5 features multiplied by 4 variables).

For the training task we arranged the training data so that it fits the NN architecture shown in Figure 2. Namely each feature vector has the structure $x = [x^{(1)}, x^{(2)}, \ldots x^{(10)}]$, where $x^{(i)}$ is the feature vector corresponding to i^{th} mass value in the set of ten considered mass values. Hence, we have 2000 training samples, where each sample is a 10×20 matrix. The NN shown in Figure 2 was implemented using the PyTorch platform (Paszke et al., 2017). The identical NNs in the first layer have three hidden layers of size 5 with a `tanh` activation function and a softmax function as last layer. We used the Adam optimization algorithm for learning the parameters of the NN with default values for the hyper-parameters. We trained the parameters of the NN for 3000 iterations. After training, we extract one of ten identical NNs in layer 1 of the NN in Figure 2. This network corresponds to the map $g(x;\beta)$ defined in the problem setup section. For testing we revert to the original form of the training and test data, that is we have 20,000 training samples and 5,000 test samples, where each sample is a feature vector of dimension 20. Figure 5 depicts the confusion matrices against the training and test data when applying them on the map $g(x;\beta)$.

Next, we use training data to train a new NN with the *same* structure as $g(x;\beta)$, that is, the same number of layers and activation functions. We trained the network under the same conditions mentioned above. This setup corresponds to the case where $w_n = \frac{1}{N}$, hence we actually minimize a different

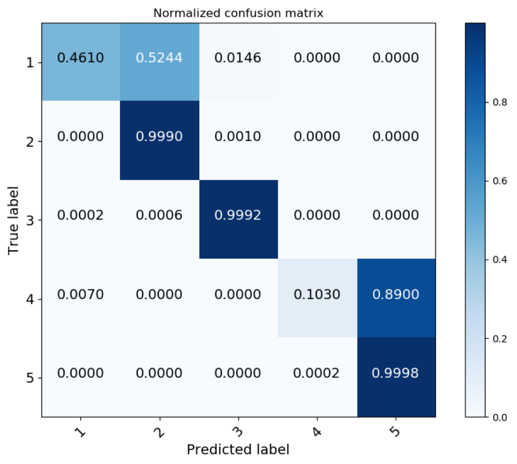


(a) Confusion matrix for the training data

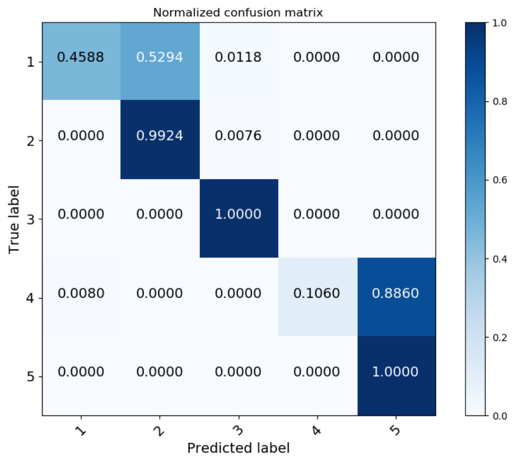


(b) Confusion matrix for the test data

Figure 5. Confusion matrices after training the UQ-based NN: (a) confusion matrix corresponding to the training data; (b) confusion matrix corresponding to the test data



(a) Confusion matrix for the training data



(b) Confusion matrix for the test data

Figure 6. Confusion matrices for a NN with a structure identical to $g(x;\beta)$: (a) confusion matrix corresponding to the training data; (b) confusion matrix corresponding to the test data

cost function. Therefore, different results are expected. The confusion matrices corresponding to this case are shown in Figure 6. The second NN performs worse than the UQ-based NN since, where the comparison is performed with respect to the values om the diagonals. This result should not come as a surprise, since we actually minimize a different cost function:

$$\mathbb{E}\left[L(\beta;x(\xi),y)\right] \approx -\frac{1}{N}\sum_{j=1}^{L+1}\sum_{n=1}^{N} y_j \log \bar{y}_j(\xi^{(n)}). \tag{17}$$

Since the initial values for the NN parameters are randomly initialized, the training results will slightly vary every time we execute a new training session. The UQ-based NN does perform better though. We do pay a price. Although the number of training parameters is the same, both the forward and backward computations are more expensive since they need to be done for N identical copies of the NN in the first layer.

6. Conclusions

We addressed the problem of learning a classification-based diagnosis using synthetic data. The data is generated by a model with uncertainties. We used concepts from the field of uncertainty quantification such as generalized chaos polynomials to represent the uncertainty in the training data. This approach reduces the number of model simulations we need to execute, as compared to a Monte-Carlo approach. We proposed a NN architecture that considers training data uncertainties. We applied our approach to learning a NN-based classifier for detecting faults in an elevator system. Under similar structural and training conditions, we demonstrated that the UQ-based NN performs better than a standard NN.

References

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.

Dozat, T. (2013). Incorporating Nesterov Momentum into Adam. *ICLR Workshop*.

Fritzson, P., & Bunus, P. (2002). Modelica – a general object-oriented language for continuous and discrete-event system modeling. In *in proceedings of the 35th annual simulation symposium* (pp. 14–18).

Holtz, M. (2008). *Sparse grid quadrature in high dimensions with applications in finance and insurance*. Springer.

Honda, T., Saund, E., Matei, I., Janssen, B., Saha, B., Bobrow, D. G., ... Lattmann, Z. (2014, August). A simulation and modeling based reliability requirements assessment methodolog. In *Proceedings of international design engineering technical conferences and computers and information in engineering conference (asme 2014)* (Vol. 7).

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*.

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

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).

Minhas, R., de Kleer, J., Matei, I., Saha, B., Janssen, B., Bobrow, D., & Kurtoglu, T. (2014). Using fault augmented modelica models for diagnostics. In *Proceedings of the 10th international modelicaconference* (pp. 437–445).

Ogura, H. (1972, September). Orthogonal functionals of the poisson process. *IEEE Trans. Inf. Theor.*, *18*(4), 473–481.

O'Hagan, A. (2013, may). *Polynomial chaos: A tutorial and critique from a statistician's perspective* (Tech. Rep.). University of Sheffield, UK.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch.

R. H. Cameron, W. T. M. (1947). The orthogonal development of non-linear functionals in series of fourier-hermite functionals. *Annals of Mathematics*, *48*(2), 385-392.

Saha, B., Honda, T., Matei, I., Saund, E., de Kleer, J., Janssen, W. C., ... Bobrow, D. G. (2014, August,). Model-based approach for optimal maintenance strategy. In *Proceedings of second european conference of the prognostics and health management society.*

Smith, R. C. (2013). *Uncertainty quantification: Theory, implementation, and applications*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Wiener, N. (1938). The Homogeneous Chaos. *American Journal of Mathematics*, *60*(4), 897–936.

Xiu, D., Lucor, D., Su, C.-H., & Em Karniadakis, G. (2003). Performance evaluation of generalized polynomial chaos. In *Computational science — iccs 2003: International conference, melbourne, australia and st. petersburg, russia, june 2–4, 2003 proceedings, part iv* (pp. 346–354). Berlin, Heidelberg: Springer Berlin Heidelberg.