

Presented at: ISA 93 (Instrument Society of America), Chicago, IL, USA, Sept. 19-24, 1993.

NEURAL NETWORKS FOR FAULT DIAGNOSIS BASED ON MODEL ERRORS OR DATA RECONCILIATION

G.M. Stanley*
Principal Scientist
Gensym Corporation
10077 Grogan's Mill Road, Suite 100
The Woodlands, TX 77380

KEYWORDS

Neural networks, Diagnostics, Error identification, Simulation

ABSTRACT

Instrument faults and equipment problems can be detected by pattern analysis tools such as neural networks. While pattern recognition alone may be used to detect problems, accuracy may be improved by "building in" knowledge of the process. When models are known, accuracy, sensitivity, training, and robustness for interpolation and extrapolation should be improved by building in process knowledge. This can be done by analyzing the patterns of model errors, or the patterns of measurement adjustments in a data reconciliation procedure. Using a simulation model, faults are hypothesized, during "training", for later matching at run time. Each fault generates specific model deviations. When measurement standard deviations can be assumed, data reconciliation can be applied, and the measurement adjustments can be analyzed using a neural network. This approach is tested with simulation of flows and pressures in a liquid flow network. A generic, graphically-configured simulator & case-generating mechanism simplified case generation.

NEURAL NETWORKS FOR FAULT DIAGNOSIS BASED ON MODEL ERRORS OR DATA RECONCILIATION

Neural Networks

In the systems studied here, the run-time job of the neural network is to detect and diagnose faults. Inputs, or "features", are presented to the network. There is one output for each possible fault, and one for "normal" operation. The output with the highest value is considered the most likely fault. This is a "classification problem".

Prior to run-time use, there is a training phase. Here, sets of training data are presented to the network. Each training case includes the feature inputs, and also the desired outputs. In the classification problem considered here, the desired output corresponds to the known class (fault or normal operation) for that case.

Neural networks have strengths for modeling systems and solving classification problems like fault detection & diagnosis, especially in nonlinear systems. [Kramer & Leonard, 1990]. The neural net learns the plant model based solely on the data. This approach has limitations, however, when extrapolation beyond the training data is required. Even "interpolation" in between training points can be risky if the data has been "overfitted" to a small training set not completely representative of the overall feature space.

Choosing the overall architecture, including the right number of nodes in a network is still somewhat of an art. If you have an inadequate number of nodes, you cannot adequately represent the plant to the desired accuracy. However, it is important to limit the number of nodes in a conventional neural network, to force "generalization" so that inputs between the data points generate a reasonably smooth network output as the inputs change.

In principal, neural networks can represent arbitrary, multivariable functions to any degree of accuracy, for use in functional approximation and in classification. This might make any preprocessing of data seem superfluous. However, in practice, there are various problems associated with selecting the best network architecture, and with training. People have commonly found it necessary to "help" the network by carefully choosing their feature set, and doing some preprocessing to better extract the features they are really looking for. This paper explores a formal method for making use of engineering models.

Combining neural net and engineering model-based technologies.

Model-based approaches can make effective use of simple models such as material, energy, and pressure balances, which cover a wide range of conditions, both normal and extreme cases due to failures. The models are available from years of engineering experience. Intuitively, one would expect this information to be useful. These provide "generalization" over a wide range of conditions, with well-known and understood limits.

Most models have some parameters that must be "learned". Or, in the case of equations such as valve pressure drop, there are the inevitable model errors. A hybrid system should be able to use the form of the models, but include a neural network learning component.

The hope of hybrid systems is to reduce the brittleness of neural networks when extrapolating beyond the training data, and to improve performance "inside" the training data. The models force conformance to physical laws, or highlight deviations from those laws, providing the needed generalization.

The hybrid system approach taken in this paper uses engineering model knowledge to preprocess data, generating calculated "features" for presentation to the network. The raw data is processed to generate features which take the process models into account. There are at least several possible approaches:

- (1) Generate model "residuals" (deviations) as features
- (2) Perform data reconciliation on the inputs, and use the measurement corrections as features. (In the case of dynamic models, a Kalman filter could be used analogously).

Data Reconciliation

The fundamental idea of data reconciliation is to combine information from two sources: measurements and models [Mah, Stanley, & Downing, 1976]. The models are steady-state - a set of algebraic

equations. The measurements by themselves are generally not completely consistent with the models, due to random noise, systematic (bias) errors, and modeling errors. In data reconciliation, the measurements are corrected the minimal amount possible until their adjusted values are consistent with models. For instance, the estimate of flow into a pipe tee will be forced equal to the total of the estimated flows out of the tee, even though the measurements might indicate a loss or gain of material. The size of the adjustment depends on the assumed standard deviations of the measurements. That is, a generally-accurate sensor will be adjusted less than one known to be less accurate. The data reconciliation problem can be formulated as a nonlinear optimization problem - minimizing the sum of squared deviations of the measurement adjustments, subject to the algebraic constraints. In the linear case, explicit solutions are available.

Roughly speaking, data reconciliation is the static analog of the Kalman filter, which is based on dynamic models. Unlike the Kalman filter, the standard Data Reconciliation approach is to assume perfect models. This works best for the "simple" models that are known to be good, such as material, energy, and pressure balances. For models with more uncertainty, such as kinetic models, it may make more sense to include a quadratic penalty term in the optimization rather than force exact constraint satisfaction, following the analogy to the Kalman filter approach of "model uncertainty".

Generating features for pattern analysis by the neural net

The basic strategy is to hypothesize a fault (or normal operation) and try to predict the effects - determine a "signature" or pattern of measurements unique to that fault. With a simulation, you can explore a wide range of faults difficult or impossible to safely test in a plant.

As an example of generating fault patterns, consider a single, long steady-state pipeline with a series of 6 flow measurements, all with the same design standard deviation. In normal operation, the flow sensors should all give roughly the same reading. The statistically-best estimates of the flows (under mild assumptions) are obtained by averaging the flow sensor values. Data reconciliation comes to the same intuitive result.

Now, suppose one of the sensors has a very high bias, so that the reading is too high. How do we recognize this? One approach is to simply compare to the adjacent meters. If the meter is significantly higher than the adjacent meters, this is an easily-recognized pattern of failure. To approach this systematically, you write mass balances.

If you think of writing 5 independent mass balances around the sections of pipe between the meters, you would say that theoretically the flow_in equals the flow_out. Or, at normal, no-leak conditions, the "residual" of the material balance equation

$$\text{residual} = \text{flow_in} - \text{flow_out} = 0$$

is indeed zero at normal conditions for each of the 5 mass balances. Due to measurement errors, the residuals will not be exactly zero. However, the pattern of normal errors, as an array of these 5 residuals, would still be close to (0,0,0,0,0).

By "close", we mean within the normal range based on the standard deviation of the difference of two flow measurements. Some other approximate examples follow: in all cases, b is assumed much greater

than normal variations. The patterns that follow put the 5 material balance residuals into an array, or "pattern":

(0, 0, 0,0,0): normal operation

(-b, b,0,0,0): second sensor biased high by an amount b

(0, -b,b,0,0): third sensor fails high by an amount b

(0,-b, 0,0,0): leak of magnitude b occurs between the second and third meters

These model residuals can be fed into the network as features. The network needs to determine which class (normal or failure mode) most closely matches the observed pattern of features, based on training.

Note that an advantage of using residuals as features is that the equations are known to be good for all conditions - flows of all magnitudes and signs. If the neural network were just trained on the 6 "raw" measurements, you have no guarantee that the network has really "generalized" and "learned" the concept of material balance. If the training covers a limited range, it may have to extrapolate later. If the training covers a wide range, the result still may not be as accurate as the actual balance. A loss of accuracy would mean less sensitivity in detecting faults, and a higher probability of misclassification.

In a conventional feedforward network with linear and sigmoidal nodes, if you train a net based on process data only, you will likely only learn part of the range of all possible conditions. Extrapolation may be dangerous. In the case of radial basis functions, the net will "tell" you that the data is outside of its range of training data, so you may just lose the ability to make a decision. You may also be unable to obtain results if new data falls in a large gap between training data. With either architecture, a hybrid using residual calculations can offer more - as long as the range of model validity is greater than the training data set.

Why then further consider the output of data reconciliation measurement adjustments? One possible answer is that the data reconciliation has already "learned" the interactions between each measurement and the estimates of all the other variables, not just the adjacent ones.

The measurement adjustments made in data reconciliation depend on the hypothesis that there are no unmodeled faults, such as instrument biases or leaks, just as in the earlier case of analyzing residuals. If this hypothesis is wrong, the fault will have a pronounced effect on the reconciliation, generating obvious fault patterns.

Consider again the single pipeline example, where the reconciled result is equivalent to the average value. With a high-bias failure at one sensor, the reconciled result is that the one high measurement is reduced significantly, and the others are all pulled up somewhat. This array of measurement adjustments can be thought of as a pattern. Each possible failure will result in a distinctive "signature" in the output of the data reconciliation. In the pipeline example, there are 6 measurements, so there are 6 adjustments, used as features for the network input. A high bias b on the third flow sensor would result in a pattern of measurement adjustments of :

(b/6, b/6, -(5/6) b, b/6, b/6, b/6).

Distinctive patterns of this form apply to each of the other failure modes. Note the loss of dimensionality with simple residual analysis (5 residuals), vs. retaining the full dimensionality of 6 measurement adjustments.

Contrast the reconciliation approach with the simple residual-based approach. When using residuals as features, the residuals only picked up the local effects - comparing meters against the immediately adjacent meters. There, when the second sensor had a high bias, no residual comparison was made with the fourth, fifth, or sixth sensors, even though a comparison against them would be just as valid. Those comparisons simply correspond to formulating material balances around different portions of the pipeline. Information is not fully utilized when looking only at residuals, forcing the neural network to have to learn more. The data reconciliation forces attention to the entire set of constraints. One might expect that the generalization capabilities would be better with data reconciliation.

Overall Process of building the fault diagnosis system

(1) Build a configurable simulator

(2) Use the simulator to generate cases, adding random noise to sensors

- Randomly vary some of the inputs.
- Include sensor bias cases - high and low failures of larger magnitude than the normal noise.

(3) Process sensor data using various filters, data reconciliation, equation residuals (imbalances), or other calculations

(4) Generate feature data for input to neural network

- Use sensors and valve positions as features in standard configuration
- Use measurement adjustments and valve positions in data reconciliation case
- Use residuals (option)

(4) Train the network

- Solve as a classification problem
- Experiment with various architectures (number and type of nodes/layers)
- Cross-validate architecture by dividing into testing/training data (Figure 1)
- Train on complete network once architecture is chosen.

(5) Run-time use

- Preprocess to generate features using same technique selected for training
- Output classification - take highest output

System architecture

The overall environment is G2, a general-purpose real-time development & deployment environment. G2-based procedures analyze the schematic, and automatically generate the text of equations to be compiled and solved in the IMSL/IDL mathematical package (now called Wave Advantage). The neural network training and run-time classification is done using NeurOn-line, a real-time neural network package from the makers of G2. For this work, both conventional sigmoidal and radial basis functions were used.

The object-oriented and graphical framework of G2 was very convenient for rapidly developing the application. Graphical objects are used to configure the schematic of the process for simulation, reconciliation, and feature generation. In addition, the cases and case generation specifications themselves are objects.

The system model

The type of systems modeled for this paper are liquid piping networks, such as those of a municipal water grid or plant cooling system. A simulation is based on a graphical configuration of objects such as pumps, pipes, valves, orifice meters, pipe junctions, sources or sinks. The simulation currently includes overall material balances, pressure balances, and equations representing the pressure vs. flow characteristics of pumps, valves, meters, and pipes. The simulator solves both the nonlinear equations, and linearized versions of those equations.

Measurements of flow and pressure, along with valve positions are assumed to be the only inputs available for use by data reconciliation and the neural networks studied here.

In the particular sample shown in Figure 2, the "raw" features are the 8 measurements and the 3 valve positions. The failures simulated are high and low biases for the sensors. Thus, there are 16 failure modes and 1 normal mode, for a total of 17 possible classes. The neural network thus has 17 outputs.

To generate the simulation cases for training, sample pressures and valve positions were automatically generated by a random walk. Valve positions and underlying pressures were clamped within limits. The equations were then solved. Random measurement noise was then added (uniform noise within 3 standard deviations). For a given physical case, each of the sensor bias high and low failure cases were generated. These cases were repeated with different random measurement noise.

Standard optimization techniques (BFGS) were used to solve the error minimization problem (network training). This is generally considered superior to the older methods based on gradient search and heuristics such as "momentum factors".

Results

The simulation was used to generate a variety of cases, with sensor biases as the sample faults for the studies. Data sets were generated for direct use of the measurements and valve positions as features. Similarly, feature data sets were developed using the output of a linear data reconciliation (measurement adjustments) plus the valve positions. Simple residuals were not used as features in this study, although, as noted later, the residuals were useful in detecting simple problems prior to reconciliation.

Typical runs generated 10 "real" cases, based on setting valve positions. For each of those cases, positive, negative, and zero sensor bias cases were generated, with additive random sensor noise. This was repeated twice. Thus, from 10 underlying physical cases, a total of 510 cases were generated for training. The additive random sensor noise was found to be essential, to avoid overfitting, and especially to avoid numerical instability for training large numbers of nodes. With inadequate noise, you are forced to use a smaller number of nodes to avoid near-singularity of the optimizations during the training process. (With too much noise, it takes too many cases to get good results).

This approach of taking a physical case, and generating variations for sensor bias and noise could be applied to actual plant data cases as well. Thus, from a few sets of normal operation, a wide range of sensor faults can be simulated (for sensors not used in control schemes). Adding biases to the sensor values is easy, and forces the data reconciliation model into regions that "normal" operation might not cover. (That is good).

For training, cross-validation techniques were used to help select the number of hidden nodes. For each case, a number of hidden nodes was selected. Training was done on a subset of 2/3 of the data, holding the remaining data back for testing purposes. This was repeated 4 times, and an average of the mean square error for testing and training were compared. The number of nodes was increased (always leading to reduced training error) until the point where the testing error remained constant or increased. Final training is done on the entire data set once cross-validation has been used to select the number of nodes. The typical number of hidden nodes in the 3-layer networks was 10-60, although anywhere from 5 to 160 were tried.

Scaling of the data was found to be important, both for sigmoidal nodes and radial basis function nodes. For the first set of runs, the scaling was not done, and the classification error was always too large for a useful system. (Scaling was less crucial for the runs using data reconciliation, since the adjustments were more similar in magnitude than the absolute values). In some cases, it was difficult to even get convergence of the optimizer during training. The scaling was done by forcing the means and standard deviations in a 0-1 range by both a multiplicative factor and an additive factor. The multipliers for the neural-net input data for reconciled cases ranged from about .5 to about .0004 - indicating that scale differences of 1000 can have a significant impact on the results.

A large number of "outliers", which deviate highly from assumptions, were found to reduce the network classification accuracy. Some of the first networks trained on the simulation data performed badly because of a significant number of unnoticed glitches in the simulation/reconciliation (due to non-convergence). This was not investigated further once an outlier detection strategy was included with the simulation. Normally, a few outliers may just lead to an increase in the number of required nodes, but they otherwise do not cause much of a problem.

To detect outliers, the equation residuals were checked for the raw data. Note that when only measurement adjustments are used as features, it is not so obvious when you have an outlier. The effect of Data Reconciliation is to "smear" errors by propagating them through all the constraints. This is well-known and good under normal, low-noise conditions, and useful mathematically under fault conditions as we have noted, but means that simple limit checks may not detect the problems. Occasional numerical glitches in the simulation or reconciliation had to be intercepted before they affected the feature data sets for training. These glitches were detected for an entire case by calculating the residuals of all the constraints. These glitches also could also have been caught by simple limit checks. When training a net with live process data, you would have to perform the same checks.

The nets using radial basis functions trained much faster than their sigmoidal counterparts, for the same number of nodes (about a factor of 60 for the nets considered). In addition, radial basis functions have the advantage of providing their own built-in error analysis - extrapolation can be recognized and avoided, unlike conventional sigmoidal nets which can unknowingly give meaningless results due to

extrapolation. The needed extrapolation is done by the upfront model analysis instead of the networks. Thus, sigmoidal networks were abandoned fairly early in these studies.

For systems where the bias errors introduced were close to the normal sensor noise levels, it was difficult to train a (radial basis function) network using reconciled inputs. There was apparently too much overlap of the classes for adequate clustering. In that case, the direct features worked better. Presumably this difficulty could be overcome with an extremely large training set. The best results in those direct cases were less than 1% classification error. However, in cases where the noise was small compared to the bias errors used for training, the reconciled results were better, although never achieving better than about a 4% error rate in the cases studied. In the non-reconciled cases with small noise levels, numerical (singularity) problems occurred more often than in the reconciled cases.

One interesting feature was noted during the selection of the network architecture for the cases using reconciliation. There appeared to be a local minimum in the misclassification percentage when the number of hidden (radial basis function) nodes was approximately equal to the number of outputs. For instance, in a 9-output case, there was about a 5% misclassification error with 10 hidden nodes. That level of misclassification was not achieved again on the training set until about 60 nodes were used, although the testing set error remained fairly constant at about 4% from 20 hidden nodes upwards.

Of course, the major advantage of looking at reconciled data as feature input to neural nets was in the ability to extrapolate. Data well outside the normal range of "raw" training data could be constructed to exactly duplicate a set of reconciliation adjustments for training data, resulting in the right classification.

A major disadvantage of the approach with reconciled data is the extra time required for the reconciliation, both during training and at run time. This time has to be balanced against the potential benefit of "generalization" forced by the model, e.g. to improve extrapolation.

Several other points should be made:

- Be sure to add enough (but not too much) random noise to all simulated results. This improves the generalization capability of the nets (helping to avoid overfitting), and improves numerical stability of the training process.
- Nonlinear reconciliation, like any nonlinear optimization, introduces the possibility of convergence problems. These were encountered. Nonlinear reconciliation also can take varying amounts of time.
- While training of a neural network can have problems of robustness and convergence as well, these are resolved during training, leaving a fast, robust system for runtime use.
- There are likely to be many "hidden" constraints in a real system, not modeled explicitly as constraints in a data reconciliation procedure. The neural networks will still have to learn these constraints. For instance, while in the existing study we explicitly modeled both pressure and mass balances, we hope to study the suboptimal cases of just using reconciliation on the mass balances. The tradeoff is that the simple balances are almost always "true", and that linear reconciliation is faster and more robust than the nonlinear version.
- Comparisons still need to be made on using reconciliation vs. just presenting model residuals as features for the network (possibly with some raw data as well). This may train and run significantly faster, yet still retain most of the advantages of building in model knowledge.

REFERENCES

Kramer, M.A., and J.A. Leonard, Diagnosis Using Backpropagation Neural Networks- Analysis and Criticism, Computers Chem. Eng., Vol. 14, No.12, pp. 1323-1338, 1990.

Mah, R.S.H., G.M. Stanley, and D.M. Downing, Reconciliation and Rectification of Process Flow and Inventory Data, I&EC Process Design & Development, Vol. 15, pp. 175-183, Jan., 1976.

* The author may be contacted at :

<http://gregstanleyandassociates.com/contactinfo/contactinfo.htm>