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

Greg M. Stanley* - Gensym Corporation

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

* Contact the author at:
http://gregstanleyandassociates.com/contactinfo/contactinfo.htm

CENTRAL PROBLEM ADDRESSED

- Explore several possible mechanisms for fault detection using combinations of several technologies:

  - Neural networks

  - Traditional models, often based on first principles

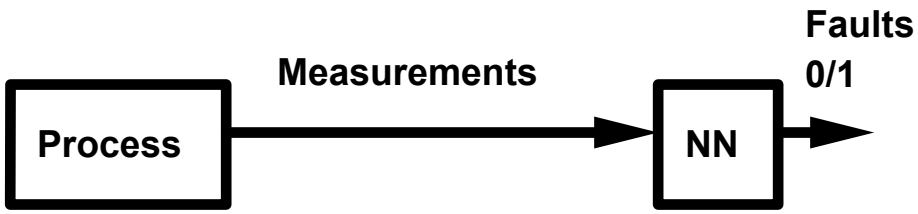
  Detect faults based on deviations from models

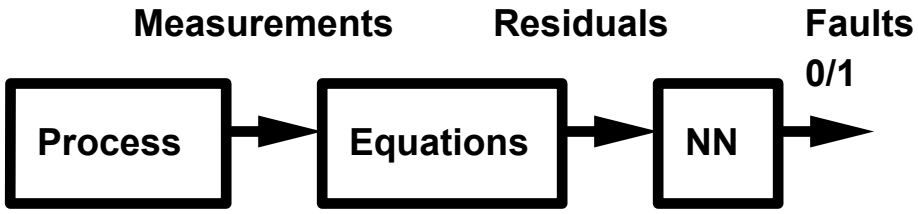
  - Data reconciliation

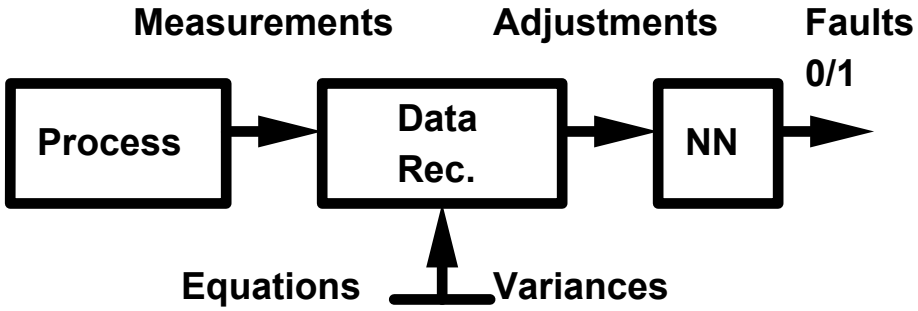
  Additional technique built upon traditional models

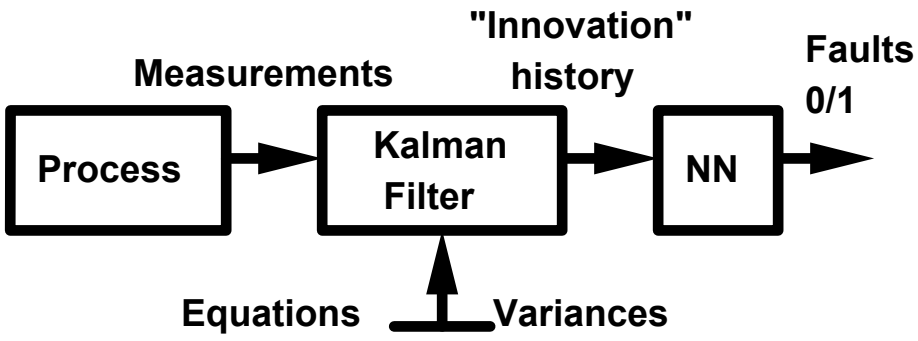
- Each technology has something to offer, and limitations
- General analysis and case study with hydraulic systems

Some fault diagnosis schemes









"Innovation" = error in measurement prediction

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

I. Neural networks background

II. Model residual analysis

III. Data Reconciliation

IV. System model

V. Results

I. NEURAL NETWORKS BACKGROUND

Neural Networks for nonlinear modeling

- Neural networks are nonlinear, multivariable models built from a set of input/output data

  - Training phase - "learn" model from the data, given pairs of input & output data arrays ("training set")

    Analogy: building regression model from data

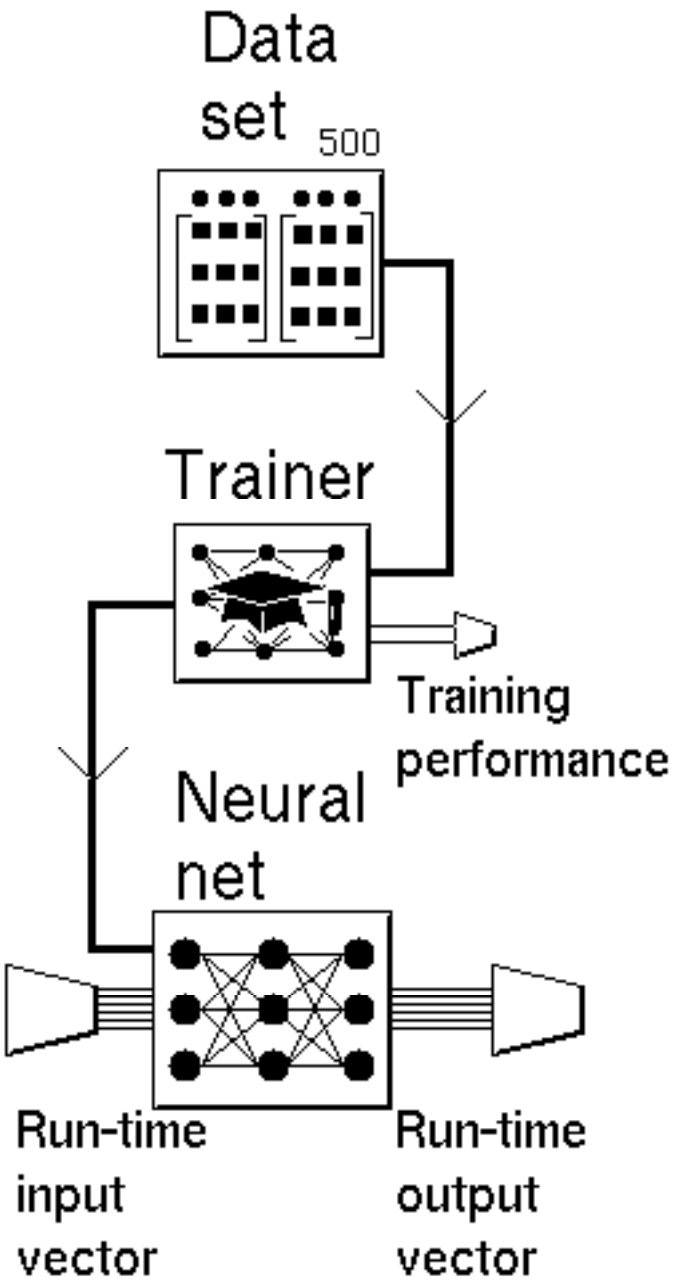
  - Run-time phase - use the model with new input array to predict the output array

    Analogy: using the regression model with new inputs

- Result is a nonlinear "black box" model

  Analogy: linear models for regression, DMC, typical controller design methods are all "black box"

Basic Neural Net elements



Neural Networks roles

- Functional approximation

  - Approximate any mapping of input to output data

  Think of it as multivariable interpolation

  - Used for interpolation, control, simulation, etc., in place of other types of models

  - Nonlinear modeling in the absence of first-principles models is a special strength of neural nets

- Classification (pattern matching)

Neural networks for classification, pattern matching, fault detection

- Input "features" are selected and collected into a vector

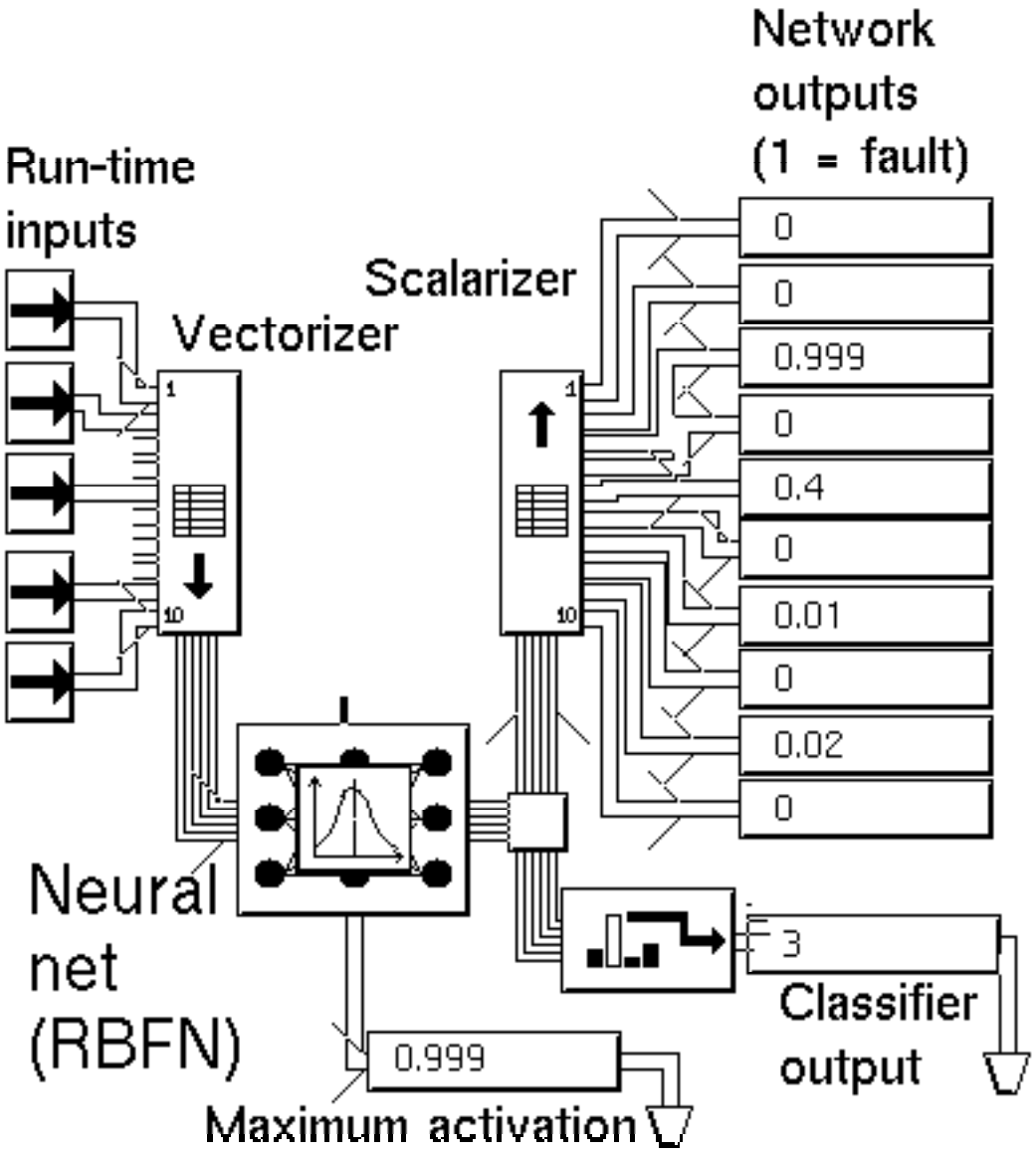
  examples: temperatures, qualities, statuses

- Each possible feature pattern belongs to exactly one of n "classes"

  example: fault detection, where "classes" are normal, fault x, fault y, ...

- There is a NN output for each of the n possible classes
- In training, 1 is applied to the correct output for the input class, 0 to the other outputs
- At runtime, a new input is presented, and an output near 1 indicates membership in that class
- A special strength of neural nets

Neural Net for classification at run time



Neural Networks vs. other techniques

- Complements traditional modelling, rule-based systems, optimization, regression, interpolation, and control
- Focus is on nonlinear systems, vs. traditional linear techniques that may be more efficient for linear systems

  - very few systems are truly linear, especially under fault or other extreme conditions

  - linearization for traditional methods often applies only within small operating regions
- First principles (simulation) models can be worked in by pre-processing or post-processing

  e.g., model the differences from first-principles models with a neural net
- Neural nets can model static or dynamic systems

  e.g., feed delayed inputs as well as current inputs

Applications areas for neural nets

- Dynamic and static process modeling
- Quality prediction & control
- Nonlinear and adaptive control
- Inferential "soft" sensing
- Fault detection and diagnosis
- Multivariable pattern recognition
- Data validation and rectification
- Time series prediction
- Process optimization
- Automated decision-making

"Backpropagation Network (BPN)"

- The "standard" network, widely-used

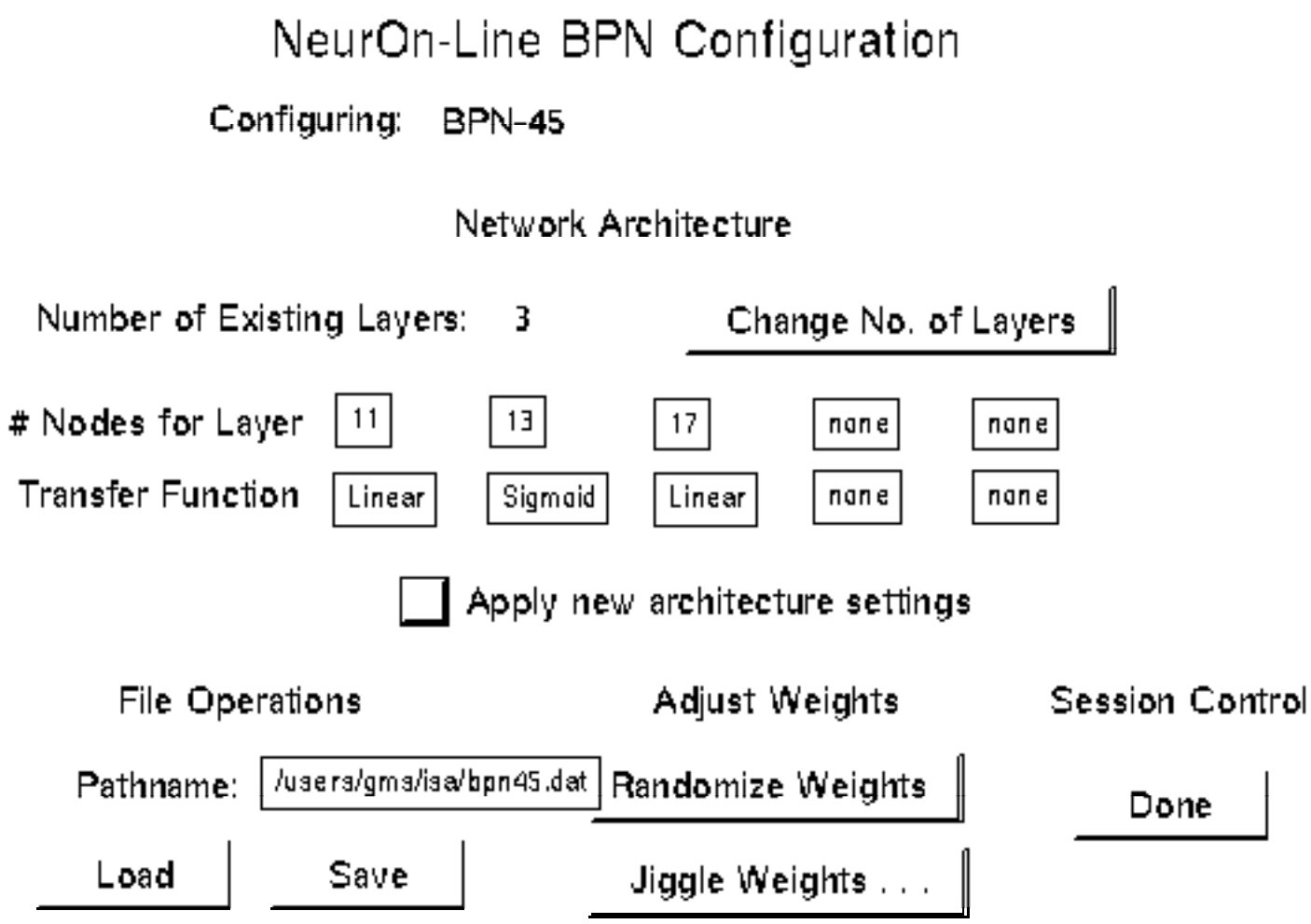
  One of 4 available in NeurOn-Line

- Named after a particular training technique

  Somewhat of a misnomer, but in common use

- Implies layered structure of nodes and connections
  - usually 3 layers (input, hidden, output)
  - "feedforward" - runtime data propagates from input through output with no feedback
- Information transmitted via connections with weights
- Each node takes weighted sum of inputs, then may apply a function to introduce nonlinearity
- Usually the nonlinear function is sigmoidal (S-shaped)

  Any NuerOn-Line layer can apply linear or sigmoidal functions


NeurOn-Line BPN Configuration
Configuring: BPN-45
Network Architecture
Number of Existing Layers: 3
Change No. of Layers
Nodes for Layer
11
13
17
none
none
Transfer Function
Linear
Sigmoid
Linear
none
none
Apply new architecture settings
File Operations
Adjust Weights
Session Control
Pathname:
/users/gma/isa/bpn45.dat
Randomize Weights
Done
Load
Save
Jiggle Weights . . .

Training and applying a neural net

(1) Choose inputs & outputs

(2) Acquire input/output training data

(3) Train the network

(4) Validate the network

(5) Apply the network

(6) Periodic retraining for adaptation

Training & applying (1) : Choose inputs & outputs

- Avoid irrelevant inputs if possible
- Functional relationship between inputs & outputs should exist
- Inputs can be calculated, model residuals, etc.

Training & applying (2): Acquire input/output training data

- Data should "cover" space of interest
  - Neural nets, like other empirical models, extrapolate poorly
  - Extrapolation may be uncovered during validation
  - Radial Basis Function nets can warn about extrapolation at run time; backpropagation nets can't
- Quality & quantity of data determine quality of result
  - Signal to noise ratio important
  - Large data sets reduce variance of predictions if a functional relationship exists
  - Validation techniques in NeurOn-Line can quantify network performance

Training & applying (3): Train the network

- Nonlinear parameter estimation
- Generally least-squares fit to training set (sum of squares of prediction errors over the data)
- NeurOn-Line uses standard optimization methods, rather than earlier backpropagation techniques - faster
- NeurOn-Line has shortcut methods for Radial Basis Function methods

NeurOn-Line Fit-Tester Configuration

Configuring: FIT-TESTER-RMS-1

Goodness of Fit Metric:

- ◆ 1. Root Mean Squared Error (functional approximation)
- ◇ 2. Fraction misclassified (classification)
- ◇ 3. Probability-based error (density estimation by rho nets)

Done

NeurOn-Line BPN-Trainer Configuration

Configuring: BPN-TRAINER-BFGS

Maximum Iterations: 100

☐ Apply new maximum from above

Training Method:

○ 1. Conjugate Gradients (Fletcher-Reeves)

○ 2: Conjugate Gradients (Polak-Ribiere)

◆ 3: BFGS (Broyden-Fletcher-Goldfarb-Shanno)

○ 4. DFP (Davidon-Fletcher-Powell)

Print Training Progress to Background Window?

○ Yes ◆ No

Done

Training & applying (4): Validate the network

- Number of parameters to be estimated by the training technique is related to the number of layers, nodes & connections
- The number of adjustable parameters (weights) must be chosen by the user or by an automated technique

  like choosing the model order in control, or the order of a polynomial in curve fitting
- Too many parameters: overfitting, no "generalization"

  like fitting quadratic polynomial to 3 points
- Too few parameters: underfitting, too much information lost

  like using a linear curve fit when quadratic is really needed
- Want to achieve the right level of generalization
- Cross-validation techniques separate "testing" data and "training" data to choose architecture

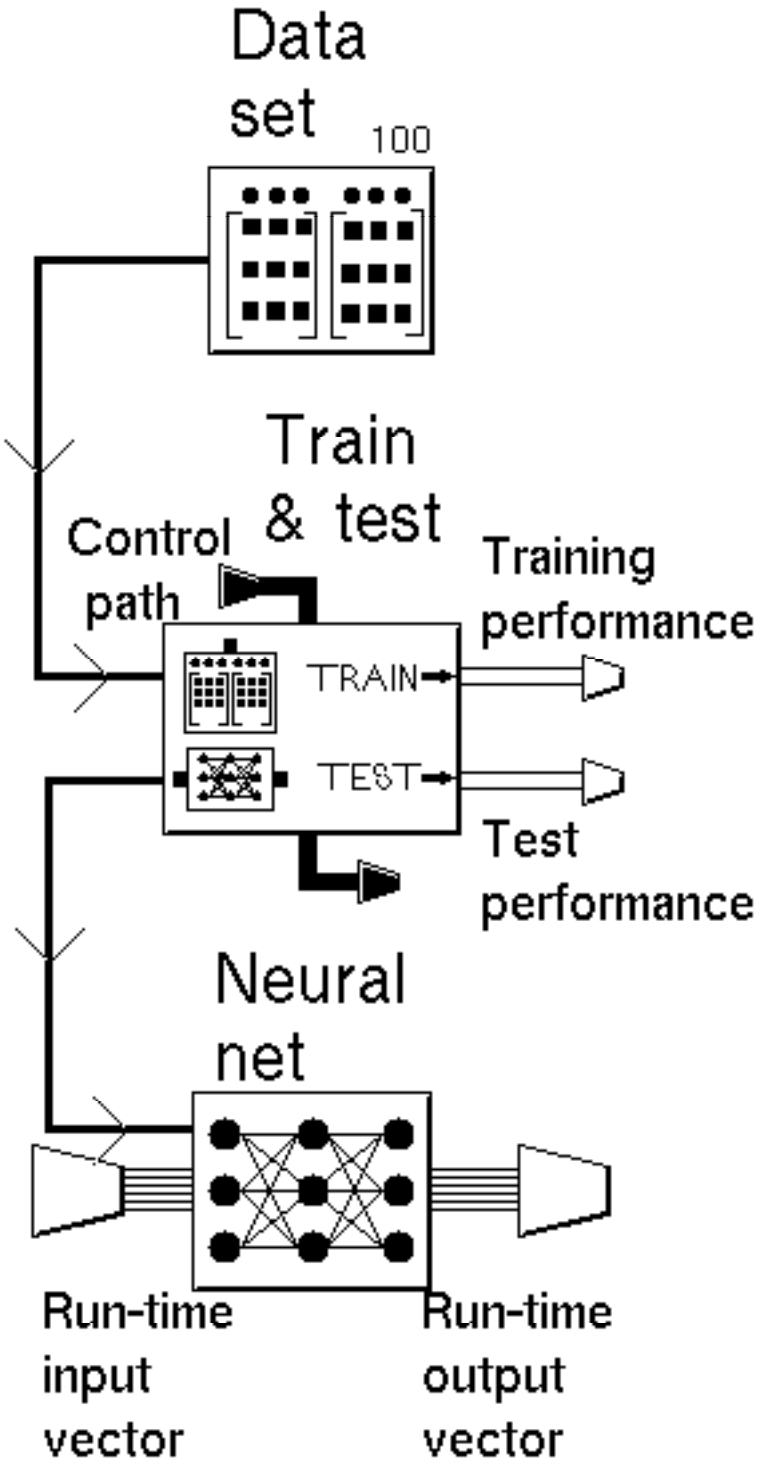
Cross-Validation

- Pick an architecture (typically, # of hidden nodes)
- Evaluate the architecture
  - Split data randomly into training and testing subsets
  - Train the network using only the training data subset - training minimizes the training error
  - Evaluate network prediction quality only over the testing subset only - "testing error"
  - Repeat multiple times with different random split of data, and average the results of the testing error

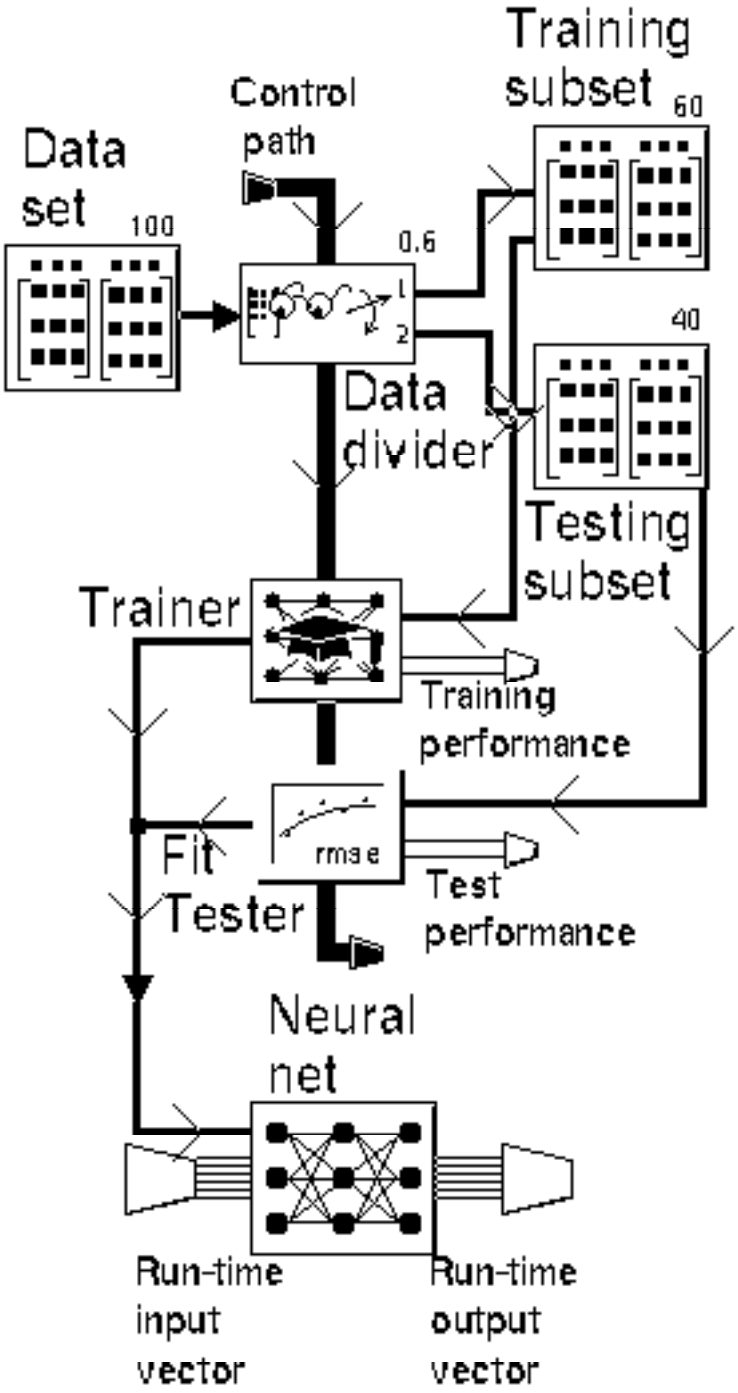
  Similar approaches exist to split the data n ways
- Repeat, choose the architecture with the lowest testing error

  Typically at a minimum between underfitting & overfitting
- Train with the final architecture, using all the data

Cross validation - high-level view



Some details of cross-validation



Training & applying (5): Apply the network & retrain as needed

- Weights, architecture fixed while running
- Cases requiring extrapolation should be flagged
- Further data acquisition & periodic retraining & adaptation
- NeurOn-Line provides support for maintaining data set
  - Adding new, novel cases
  - Forgetting old cases when newer ones are better
  - Rejecting outliers
  - Filtering or other signal pre-processing

Recognizing NeurOn-Line applications

- Difficult-to-formulate models needed for system improvement

  - Poorly-understood systems

  - Lack of experts

  - Nonlinearities

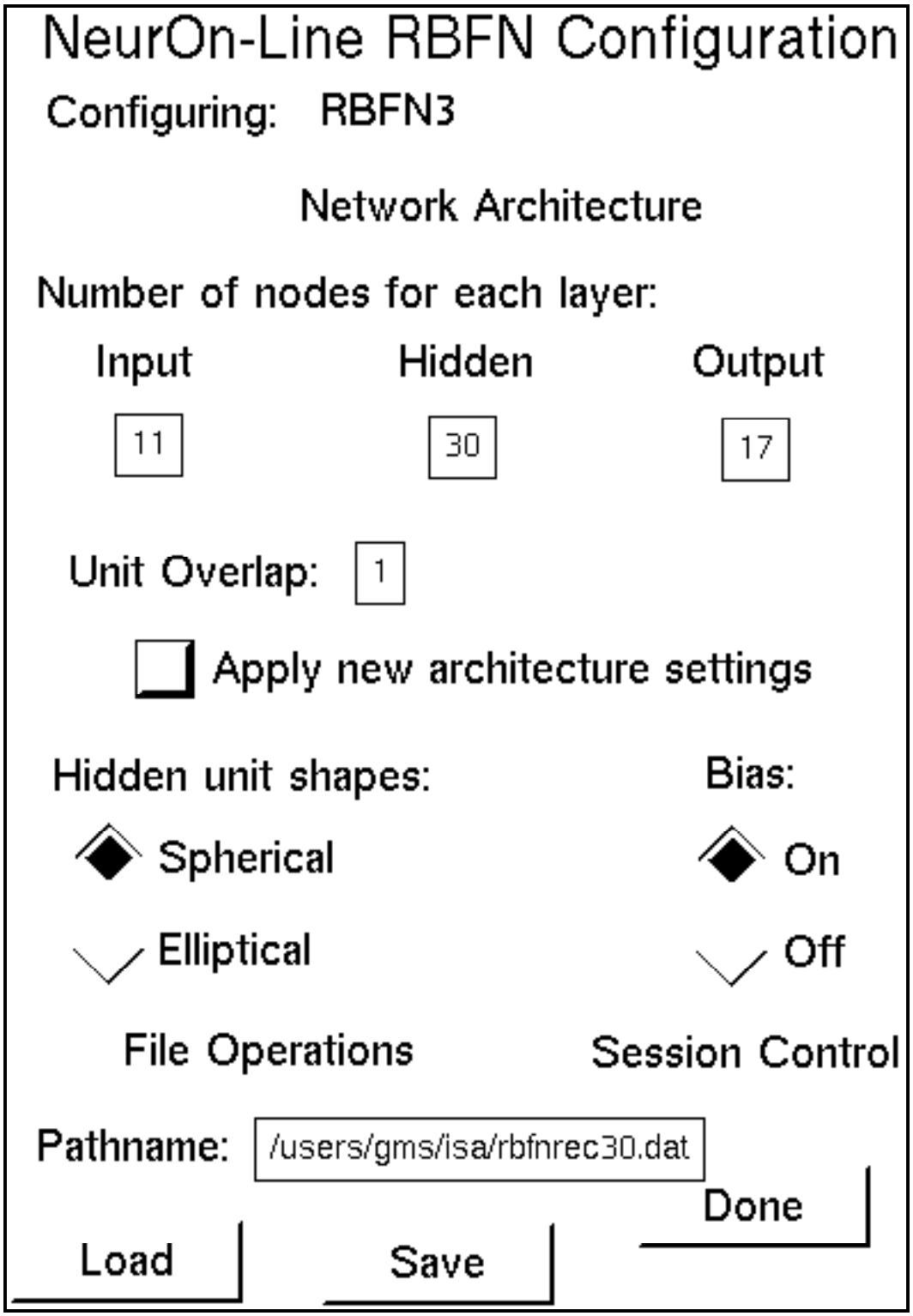
- Data available

- Functional relationship exists between inputs & outputs

- NeurOn-Line current limitations

  - Data collection, network evaluation 1 second

  - No hard limits on size, best performance for input dimension < 100, number of examples < 1000


NeurOn-Line RBFN Configuration
Configuring: RBFN3
Network Architecture
Number of nodes for each layer:
Input
Hidden
Output
11
30
17
Unit Overlap:
1
Apply new architecture settings
Hidden unit shapes:
Spherical
Elliptical
Bias:
On
Off
File Operations
Session Control
Pathname:
/users/gms/isa/rbfnrec30.dat
Done
Load
Save

NeurOn-Line

- G2-based neural network package for online applications
- Support for maintaining set of training data and building an adaptive neural network model, recognizing novelty
- Real-time pre-processing of data (filtering, feature calculations)
- Support for run-time use of the NN model
- Various network types supported

  - standard feedforward, sigmoidal

  - radial basis functions. ellipsoidal basis functions

  - Principal Component Analysis preprocessing option

  - Autoassociative nets for nonlinear principal components analysis

  - rho nets

- Training via optimization methods
- Cross-validation for testing against "overfitting"
- Graphical language for development

- GDA-based for signal processing, responding to events, sequential control

NeurOn-Line architecture

- G2 is the overall developer & end user environment
- Integrated with G2 and GDA (Gensym Diagnostic Assistant)
- Numerically-intensive training done in external C program
- Communication via remote procedure calls and file transfer

Why not just use a neural network?

- Doesn't take advantage of process knowledge

  - Network has to learn more, may generalize improperly

  - Danger of extrapolation outside of training data

  - May be difficult/time consuming to "cover" the possible operating regimes

  - A lot of testing may be required to build confidence in the network

  - Minor plant changes or operating regime changes may require extensive retraining and retesting

  - Many operating statuses change, leading to a large number of inputs to the net besides sensors

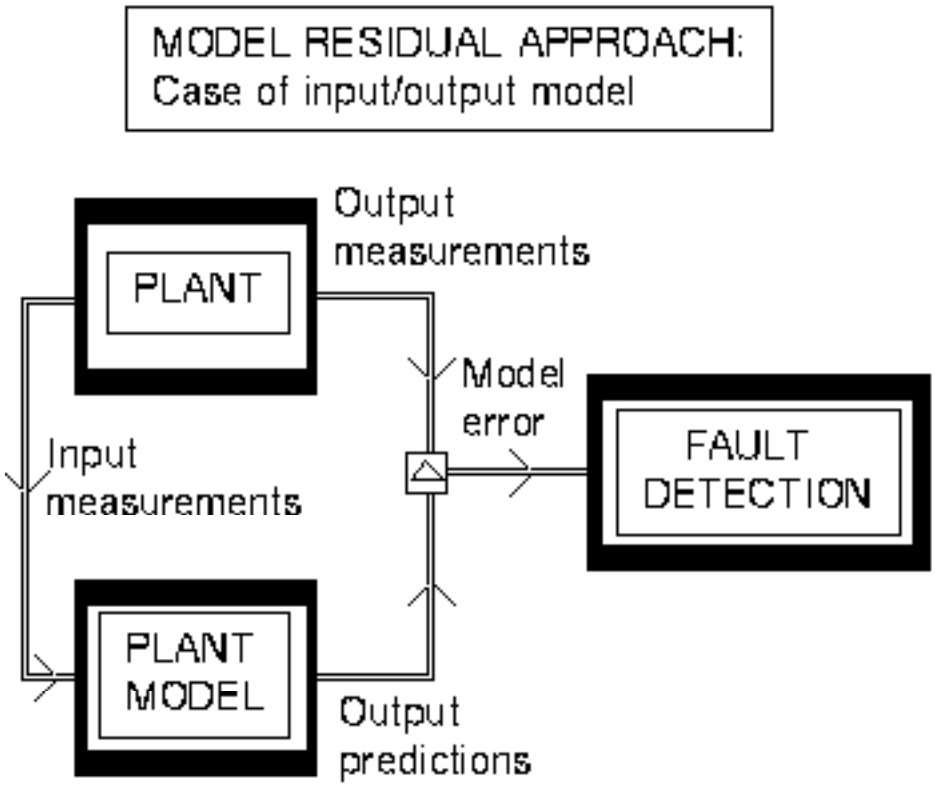
    e.g., controller statuses, parallel equipment statuses

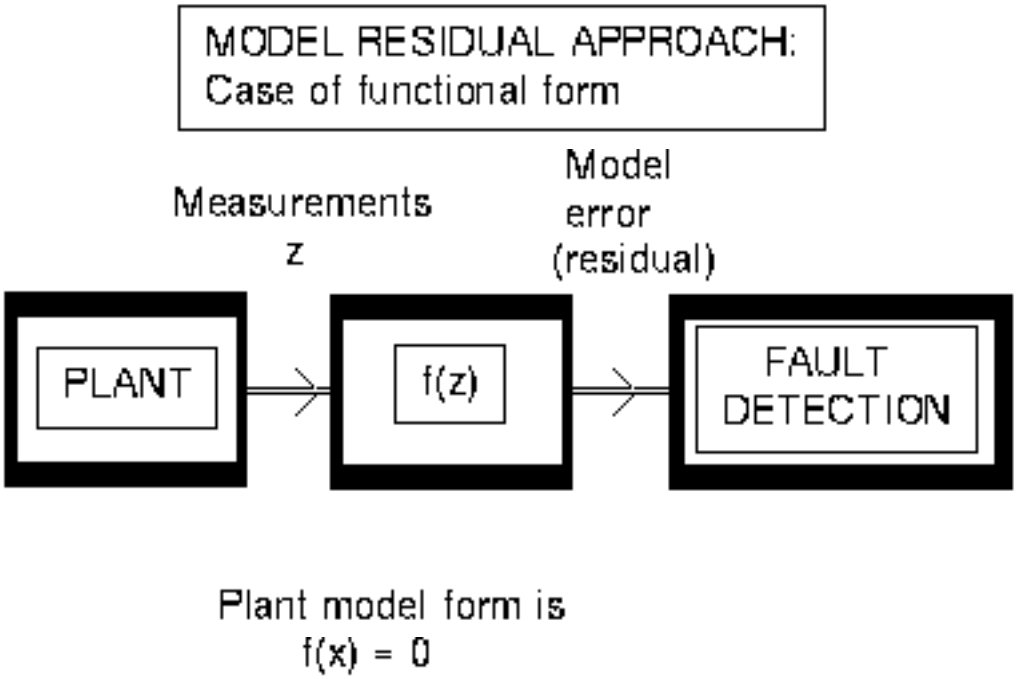
- A model or partial model with a wide range of validity may be easily available, or generated

  - Validity may go well beyond available training data

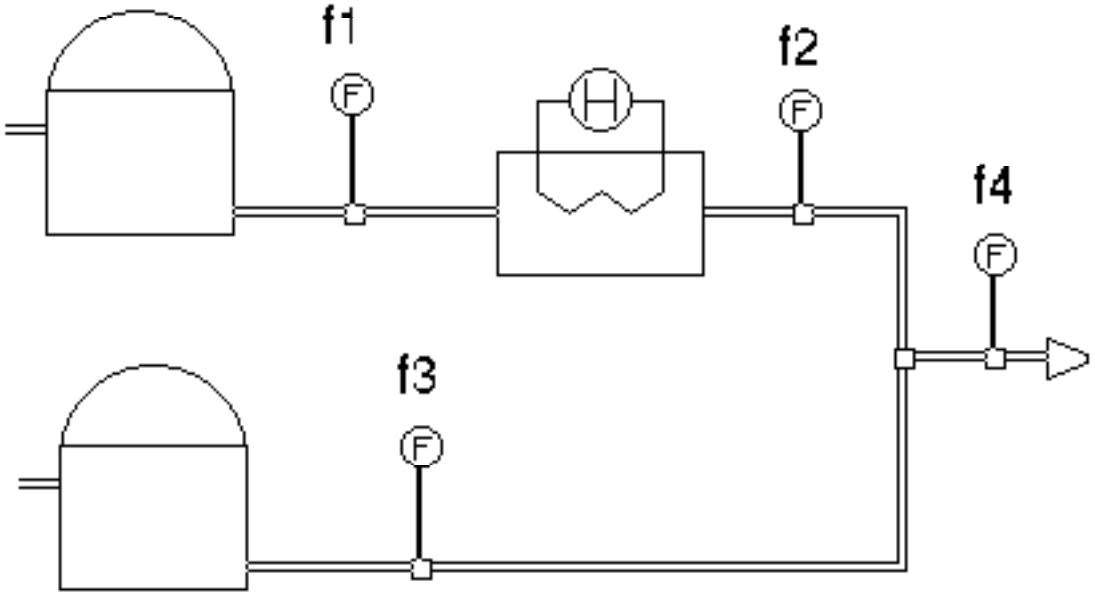
  e.g., material, energy & pressure balances, valve & pump curves, controller models

II. MODEL RESIDUAL ANALYSIS





ALGEBRAIC MODEL RESIDUAL EXAMPLE



Functional form:

$$f1 - f2 = 0$$

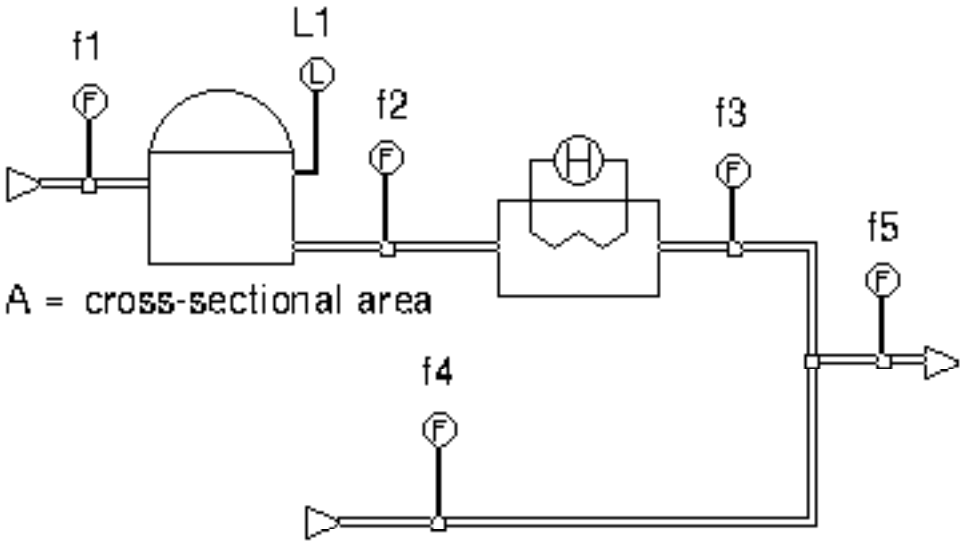
$$f2 + f3 - f4 = 0$$

Resulting residual calculations:

$$r1 = f1 - f2$$

$$r2 = f2 + f3 - f4$$

DYNAMIC MODEL RESIDUAL EXAMPLES



Functional form:	Input-output form:
A*dL/dt - f1 + f2 = 0	f2 = f1 - A * dL/dt
f2 - f3 = 0	f3 = f2
f3 + f4 - f5 = 0	f5 = f3 + f4

Resulting residual calculations to feed events:

r1 = A*dL/dt - f1 + f2

r2 = f2 - f3

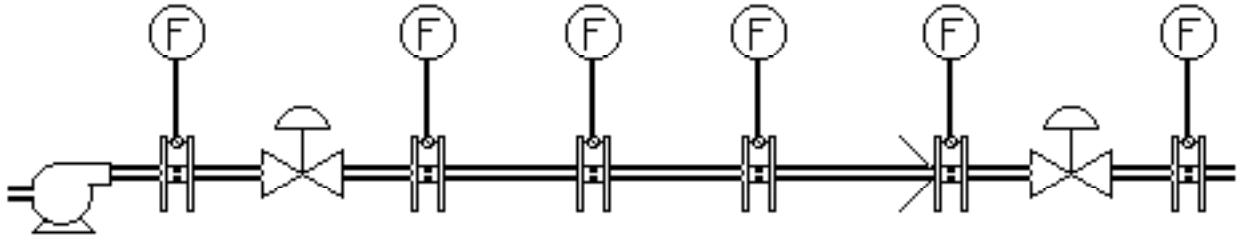
r3 = f3 + f4 - f5

(Either approximate dL/dt
or integrate equation
to solve for level change
and average or filter the flows)

No errors:

residuals: 0 0 0 0 0

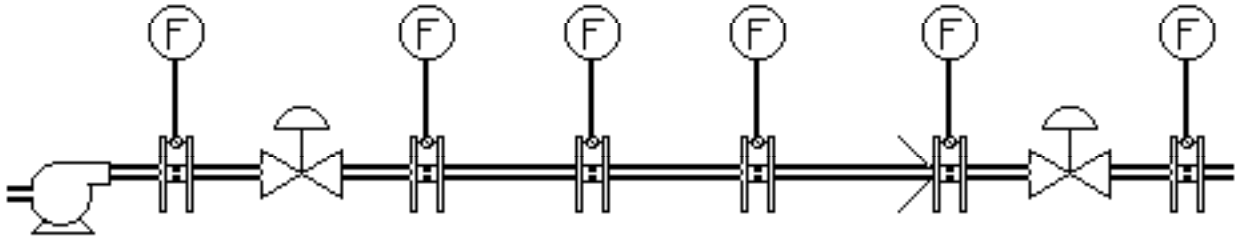
flows: 100 100 100 100 100 100



Sensor 2 biased high by 10:

residuals: -10 10 0 0 0

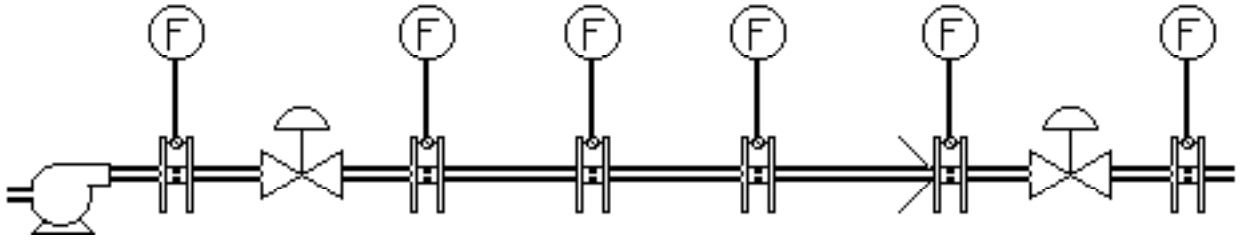
flows: 100 110 100 100 100 100



Sensor 3 biased high by 10:

residuals: 0 -10 10 0 0

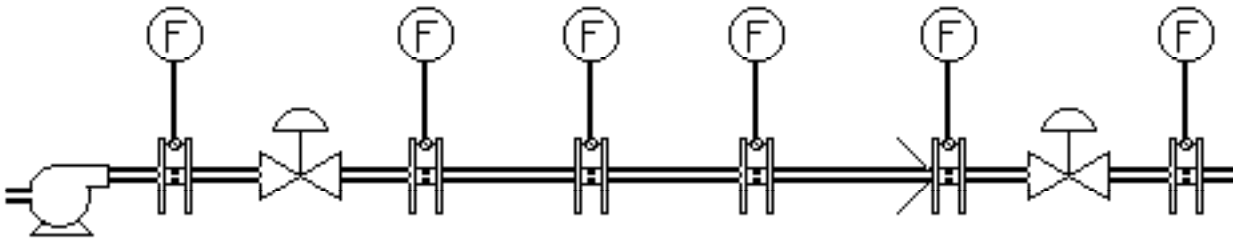
flows: 100 100 110 100 100 100



Leak of magnitude 10
between sensors 2 and 3:

residuals: 0 10 0 0 0

flows: 100 100 90 90 90 90



Model residuals form patterns for input to the NN

Residuals Fault class

(0, 0, 0, 0 , 0) : Normal operation

(-b, b, 0, 0, 0) : flow 2 biased high by amount b

(0, -b, b, 0, 0) : flow 3 biased high by amount b

(0, -b, 0, 0, 0) : leak, magnitude b, between flow & flow 3

Advantages of residuals

- Simple to compute, no iteration required, no convergence problems
- Models can be "partial", incomplete models - they are just information in the form of constraints, not a complete causal model

  Same true for data reconciliation

- Unmodeled faults will still generate residuals highlighting a fault, even though the NN will be unable to correctly classify the cause of the non-normal operation

  Same true for data reconciliation

Why not just use model residuals as NN inputs?

- Residuals are all "local" to one equation
- Residuals arbitrarily depend on which balances are chosen

  In above example, first flow is never compared to last flow, yet that is a perfectly valid comparison/balance. Only adjacent flows are compared.

- Network has to learn all the "global" interactions

  - Network may not generalize properly

- Data reconciliation fully accounts for the interactions, using ALL of the model equations, instead of just comparing adjacent sensors
- Data reconciliation allows you to specify measurement noise standard deviations, so network doesn't have to learn it

III. DATA RECONCILIATION

Data Reconciliation

- Want best estimates of variables in a system with measurements, consistent with some algebraic models

  - Combining measurement information, measurement noise properties (variances), and model information

  Analogy to Kalman Filter in dynamic systems, although usually no "process noise" is modeled, just "measurement noise"

- Traditionally associated mainly with mass & energy balances

- Associated "gross error detection" based on tests of model residuals or measurement adjustments - should be random

Data Reconciliation mainly reduces effect of instrument biases

- Uses algebraic models: steady state assumption, with a few tricks

  - Change in tank levels treated as equivalent to flow measurement

  - Other dynamic extensions exist

- Plant measurements must be averaged for time period consistent with steady-state assumption

  - Typical 4 hours - 1 day

  - High frequency noise filtered out

  - Leaves only steady state error (bias) or very low frequencies

Data Reconciliation is least-squares error minimization

- Minimize "adjustments" to raw data based on their assumed variances - weighted sum of squares of adjustments
- Minimization subject to constraint that the balances are satisfied exactly
- Nonlinear if the algebraic constraints are nonlinear

Data Reconciliation - mathematical formulation

The system

measurements: $z = h(x) + v$

constraints: $g(x) = 0$

v is the measurement noise, with covariance matrix R

R is usually diagonal

Diagonal elements are measurement variances (square of std. dev.)

The least-squares problem

Find best estimate x as solution to the problem:

minimize over x: $(z - h(x))^T R^{-1} (z - h(x))$

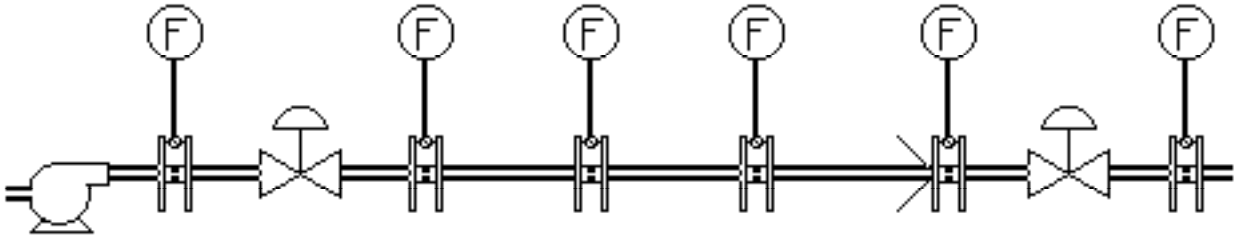
subject to: $g(x) = 0$

Special case solutions exist for linear constraints and measurements

No errors:

residuals: 0 0 0 0 0

flows: 100 100 100 100 100 100



reconciled flows:

100 100 100 100 100 100

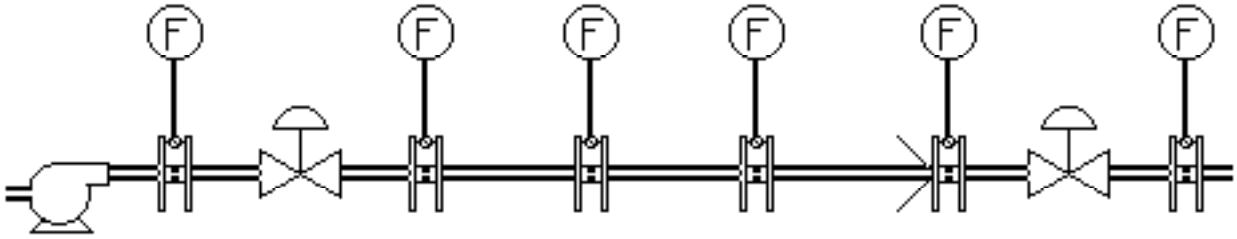
reconciliation adjustments:

0 0 0 0 0 0

Sensor 2 biased high by 12:

residuals: -12 12 0 0 0

flows: 100 112 100 100 100 100



reconciled flows:

102 102 102 102 102 102

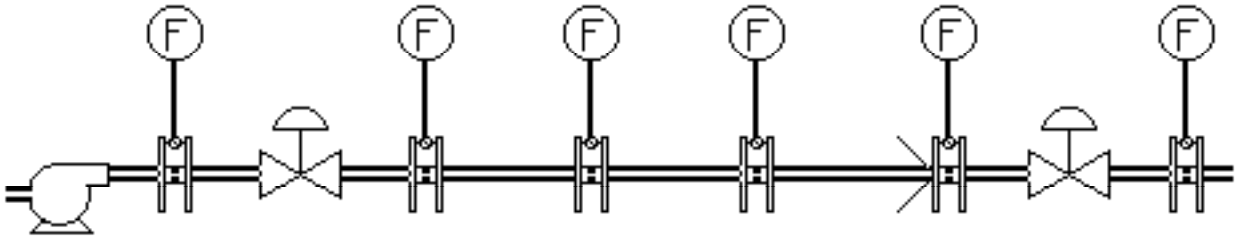
reconciliation adjustments:

2 -10 2 2 2 2

Sensor 3 biased high by 12:

residuals: 0 -12 12 0 0

flows: 100 100 112 100 100 100



reconciled flows:

102 102 102 102 102 102

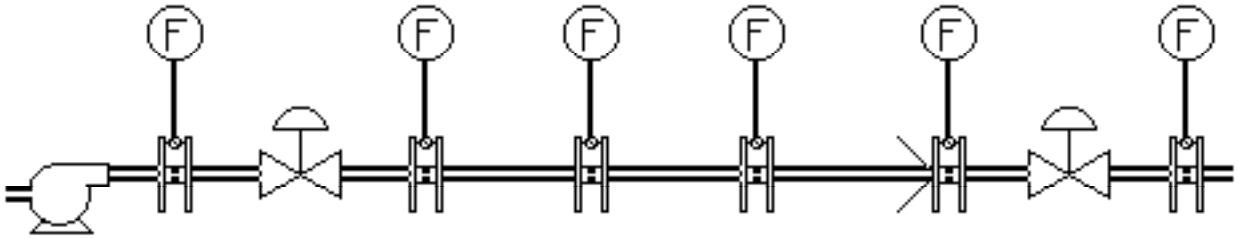
reconciliation adjustments:

2 2 -10 2 2 2

Leak of magnitude 12
between sensors 2 and 3:

residuals: 0 12 0 0 0

flows: 100 100 88 88 88 88



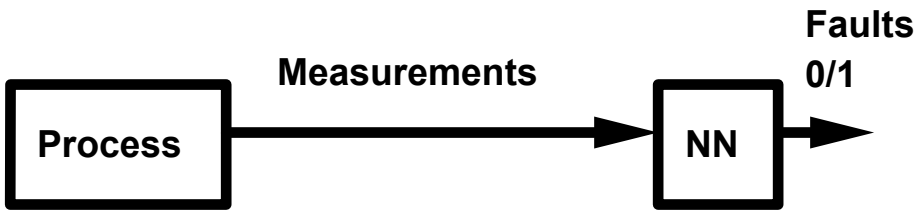
reconciled flows:

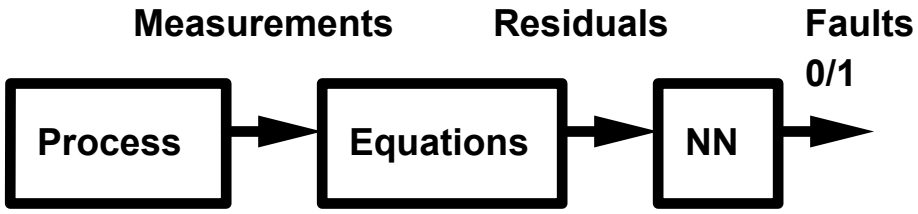
92 92 92 92 92 92

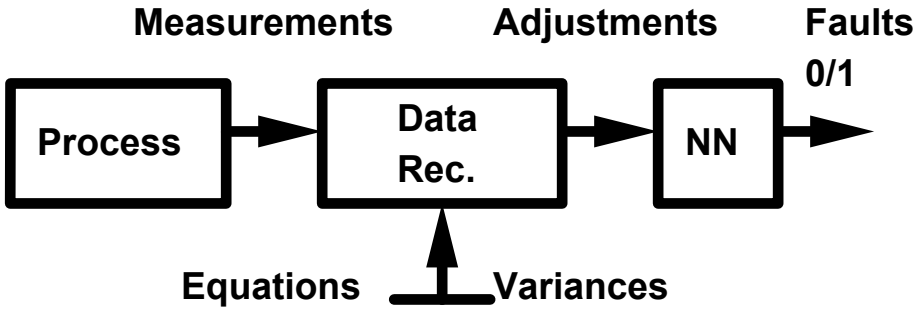
reconciliation adjustments:

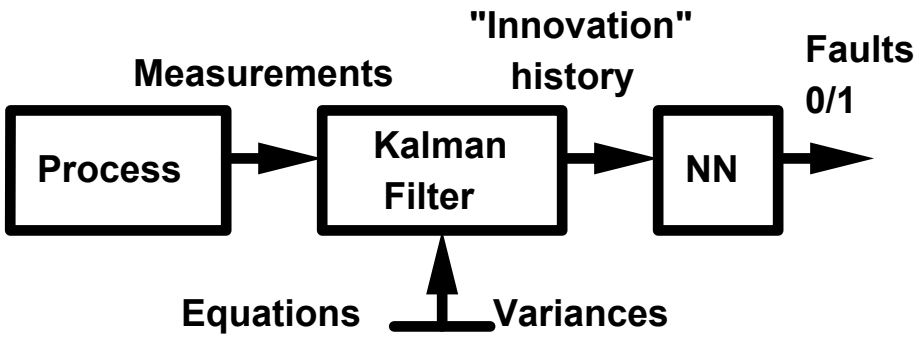
-8 -8 4 4 4 4

Some fault diagnosis schemes









"Innovation" = error in measurement prediction

IV. THE SYSTEM

Overall process of building fault diagnosis system

- Build a configurable simulator
- Select features to be used for input to the neural network
  - Sensors, valve positions
  - Model equation residuals
  - Other calculations
  - Data Reconciliation measurement adjustments
  - Filtering, averaging, other signal processing as needed
- User the simulator to generate cases - a training data set
  - Include sensor bias cases as faults
  - Add random noise to sensors
  - Randomly vary the inputs
- Train & validate the network (classification problem)
- Run-time use - use same features on real data

Overview of the water grid model

- Graphically-configured hydraulic network, as in municipal water grid
- Generation of model equations from schematic

  - Fixed pressures at sources or sinks

  - Pressure/flow models of pumps, valves, orifice meters, pipes, junctions

  - Conservation of mass

  - *Analogous to Kirchoff voltage & current laws, with device equations*

  - Generate matrices for linearization when desired
- Algebraic equations only

  - Tanks not considered, although this is a straightforward extension

The system

- G2-based schematic analyzer generates linear or nonlinear equations, sets up linear or nonlinear data reconciliation
- Equations solved by IMSL/IDL (Wave Advantage) nonlinear equation-solver
- Nonlinear data reconciliation solved by IMSL/IDL optimizers
- Case generation for NeurOn-Line (neural network)

  - G2 Generates cases of various sensor failures, simulating using above models

  - G2 outputs patterns of model residuals or data reconciliation adjustments to file for training
- NeurOn-Line does training, runs networks

IMSL/IDL (Wave Advantage) interface to G2

- Wave Advantage = IMSL/IDL, similar to MATLAB
- G2 sends commands to Wave Advantage command line interpreter as ASCII text strings - G2 looks like a user to Wave Advantage
- Optionally, G2 can generate files for compilation by Wave Advantage, triggered by command line input to Wave Advantage
- Results come back from IMSL/IDL in files

Software roles

I. G2

- Coordination of entire system
- Overall developer and user interface
- Model representation
- Schematic analyzer to generate equations from schematic
- Case generation
- Running NeurOn-Line

  Calls separate C program for training (transparent to user)

II. IMSL/IDL (now PV-WAVE)

- Solution of model equations (linear & nonlinear equation solver)
- Solution of data reconciliation optimization problem
- Specialized 3D plots for visualization

V. RESULTS

Case studies

- "Raw" features were 8 measurements, 3 valve positions
- Failures simulated were high & low biases for sensors
- Thus, 16 failure modes plus 1 normal mode - 17 classes
- Sample pressures & valve positions automatically generated
- Random measurement noise - uniform within 3 std. dev.

Conclusions

- Noise useful to force generalization, avoid numerical problems, avoid having to use small # nodes
- Too much noise harmful - need too many cases
- Cross validation would be essential in any NN application
- Scaling data important (scaling block does this automatically)
- Large number of outliers reduce classification accuracy, but a few only lead to excess, useless nodes
- Remember that some simulators can fail to converge sometimes, leading to outliers
- During case generation, check for outliers with equation residuals (outliers not obvious with reconciled data due to smearing, without more elaborate multivariate statistical tests)
- Data reconciliation step adds complexity, computing time
- Radial Basis Function nets (RBFN) train faster

- RBFN have their own built-in error analysis to avoid extrapolation
- Models themselves handle extrapolation which NN couldn't be trusted to handle - (residual or Data Rec. approach)
- Hard to train RBFN with reconciled data and small biases (vs. noise), probably due to overlap of classes in clustering step
- When the sensor noise is small vs. biases:

  - Reconciled data worked better

  - numerical problems occurred more with non-reconciled cases
- Either model-based technique has the major advantage of extrapolating beyond training data, and better results for a given number of cases