Constructing a Non-Linear Model with Neural Networks for Workload Characterization

Richard M. Yoo	Georgia Tech
Han Lee	Intel Corporation
Kingsum Chow	Intel Corporation
Hsien-Hsin S. Lee	Georgia Tech

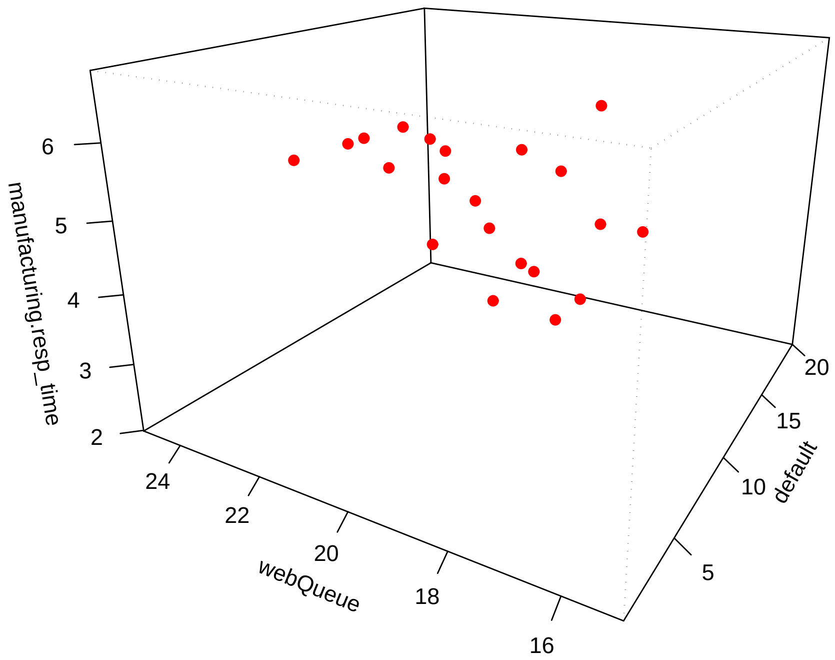
Java Middleware Tuning

- Workload tuning
  - Finding the best workload configuration that brings about the best workload performance
  - *configuration parameters*: things we have control over
    - thread pool size, JVM heap size, injection rate, etc.
  - *performance indicators*: workload behavior in response to configurations
    - response time, throughput, etc.

- Java middleware tuning
  - Inherently complicated due to its *nonlinearity*

Nonlinear Workload Behavior

- The performance of a workload does not necessarily improve or degrade in a linear fashion in response to a linear adjustment in its configuration parameters
  - Hard to predict the performance change with respect to configuration changes
  - Lottery



Sample data distribution from a case study

Nonlinear Behavior in Java Middleware

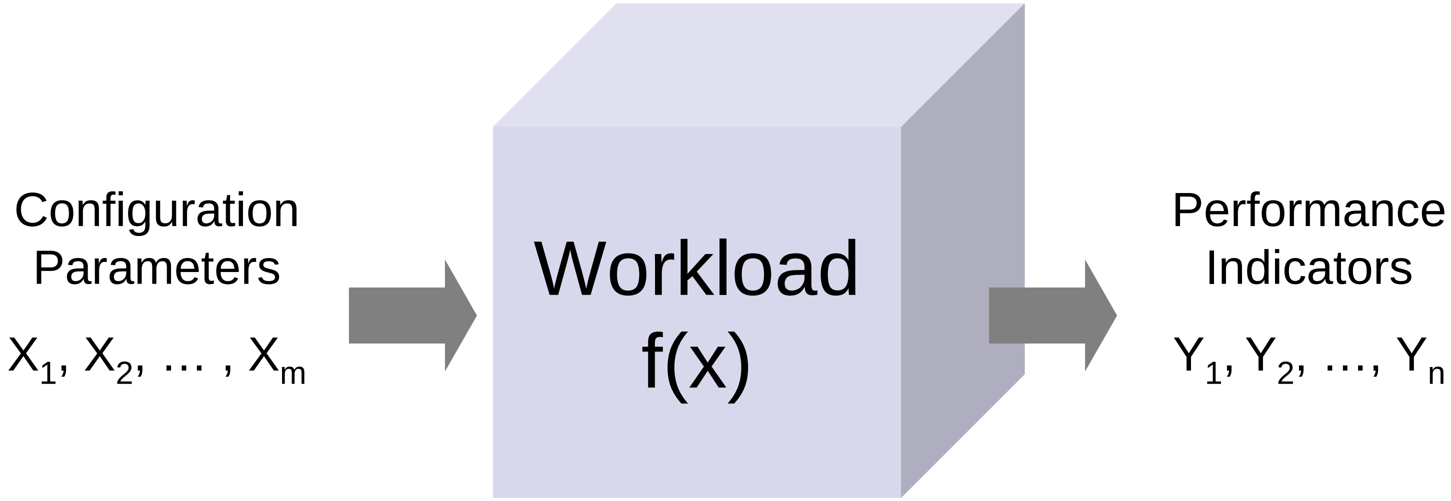
- Dominant in Java middleware behavior due to its stacked execution environment

Application
Java Application Server
Java Virtual Machine
Operating System
Hardware

A stacked execution environment

A Model Based Approach

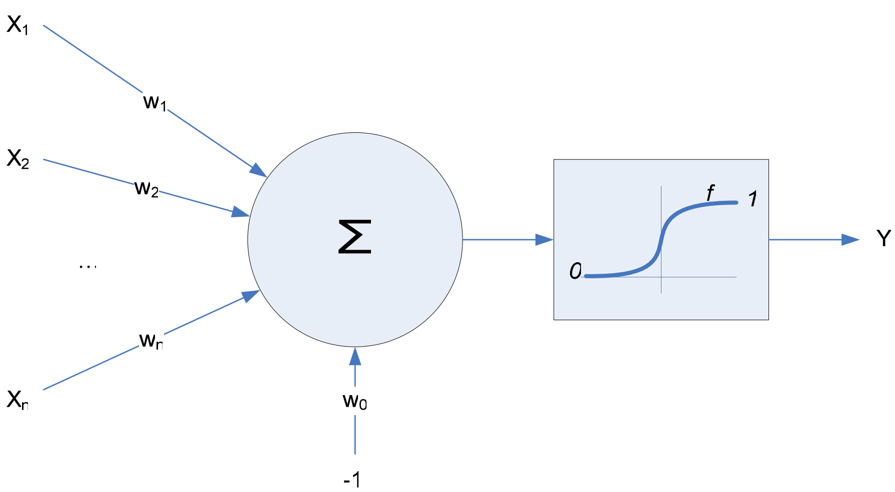
- Regard the relation between the *m* configuration parameters and the *n* performance indicators as an *m* -> *n* nonlinear function



- Map the workload tuning problem to a nonlinear function approximation problem

Function Approximation with Neural Networks

- Artificial Neural Networks
  - A network of many computational elements called *perceptrons*
  - Weighted sum of inputs + nonlinear activation function
  - Learn the input by adjusting the weights to minimize the prediction error for Y
  - Depending on the structure and organization of perceptrons, many neural networks exist



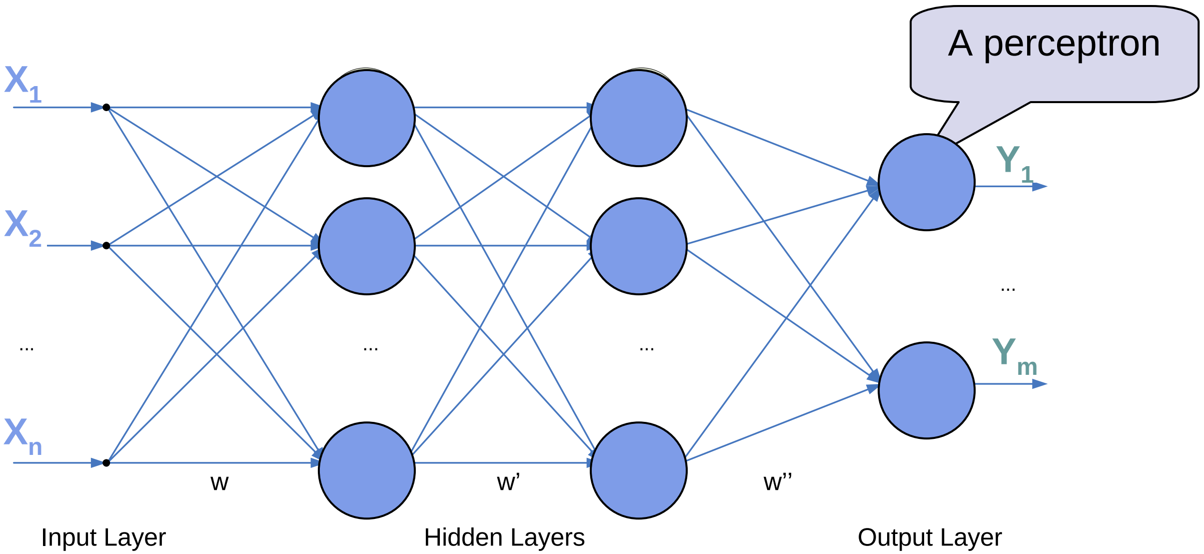
$$y = f\left(\sum_{i=1}^{n} w_i x_i - w_0\right)$$

$$f(x) = \frac{1}{1+\exp(ax)}$$

A typical structure of a perceptron

Multi-Layer Perceptrons (MLPs)

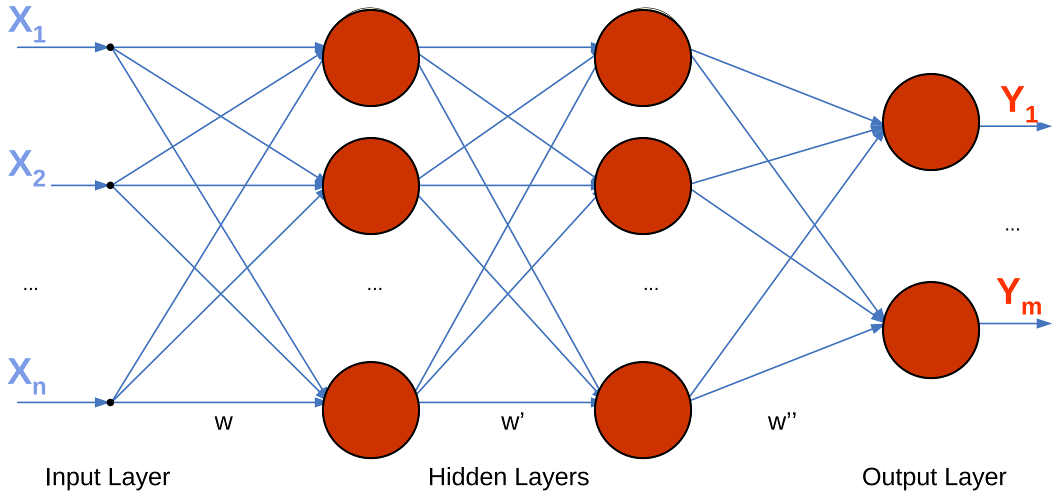
- A stacked layer of multiple perceptrons
- A feed-forward network
  - Output from previous layer feeds the next layer



A 3-layer perceptron

Training MLPs

- Backpropagation algorithm
  - By far the most popular method (standard)
  - Propagate the error of outer layer back to inner layer (blaming)
  - Each layer calculates its local error that contributed to the outer layer's error
  - Adjust each layer's weight to minimize the local error



A 3-layer perceptron

Reason for Choosing MLP

- Among many neural network configurations,
  - MLPs excel in function approximation
    - Can approximate any nonlinear function
  - MLPs are widely used in function approximation and pattern classification area

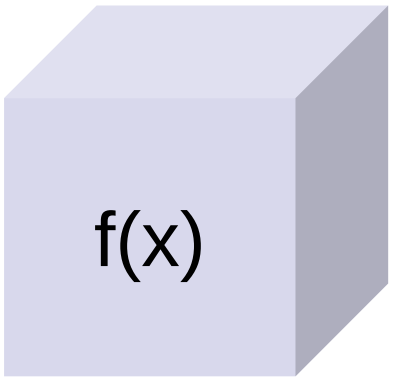
Training the Neural Network

- Neural networks are trained with samples
- Each sample is a tuple comprised of configuration parameter settings and the corresponding performance indicator values

$$(X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n)$$

- Present each performance sample to the neural network multiple times

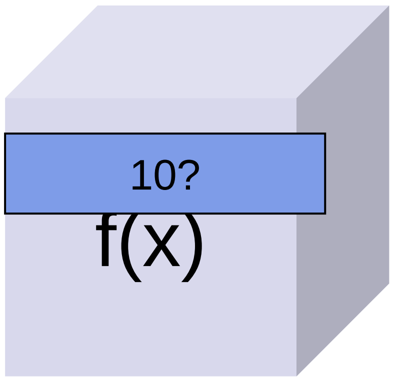
X_1 Thread pool size	X_2 JVM heap size	Y_1 Response time
10	256	13
12	256	10
10	512	9
12	512	7



Training the Neural Network

- When presented with each samples, based on the previous knowledge, neural network tries to predict the performance indicator Y'_1, Y'_2, …, Y'_n by observing given configuration settings X_1, X_2, … , X_m

X_1 Thread pool size	X_2 JVM heap size	Y_1 Response time
10	256	13
12	256	10
10	512	9
12	512	7



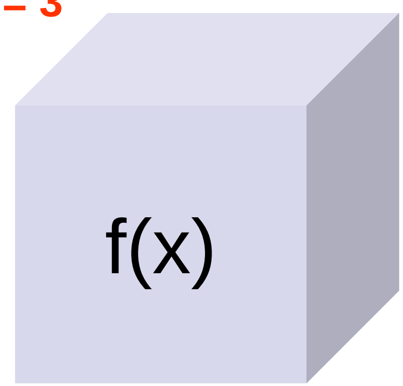
Training the Neural Network

- At the same time, neural network learns the samples by minimizing the error between predicted performance values (Y'_1, Y'_2, …, Y'_n) and the actual performance values (Y_1, Y_2, …, Y_n)

10	256	10?

error = 3

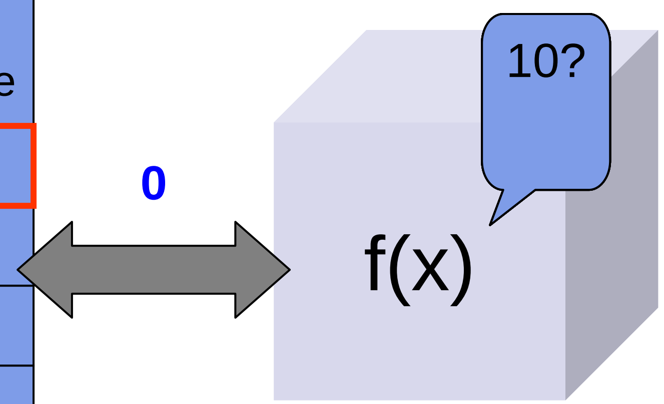
X_1 Thread pool size	X_2 JVM heap size	Y_1 Response tim
10	256	13
12	256	10
10	512	9
12	512	7



Training the Neural Network

- Process repeats over the entire samples, multiple times
- Training stops when a desired minimum error bound is reached

X_1 Thread pool size	X_2 JVM heap size	Y_1 Response time
10	256	13
12	256	10
10	512	9
12	512	7



Model Validation

- Model validity = predictability over unseen samples
  - Quantify the model validity by prediction accuracy over unseen samples
- K-fold cross validation
  - Guarantee that the sample set represents the entire sample space

```
divide the samples into k sets;
for (i in 1:k) {
   leave 1 set out;
   model.train( k – 1 sets);
   error[i] = model.error( 1 set that
   was left out);
}
average the error[];
```

Summary of Model Construction

1. Collect performance samples with varying configurations
2. Train neural network with samples
3. Perform k-fold cross validation to validate the model

Workload

- J2EE 3-tier web service, modeling the transactions among a manufacturing company, its clients, and suppliers
- 4 configuration parameters
  - Thread count assigned to *mfg queue*
  - Thread count assigned to *web queue*
  - Thread count assigned to *default queue*
  - *Injection rate*
- 5 performance indicators
  - *Manufacturing response time*
  - *Dealer purchase response time*
  - *Dealer manage response time*
  - *Dealer browse autos response time*
  - *Throughput*

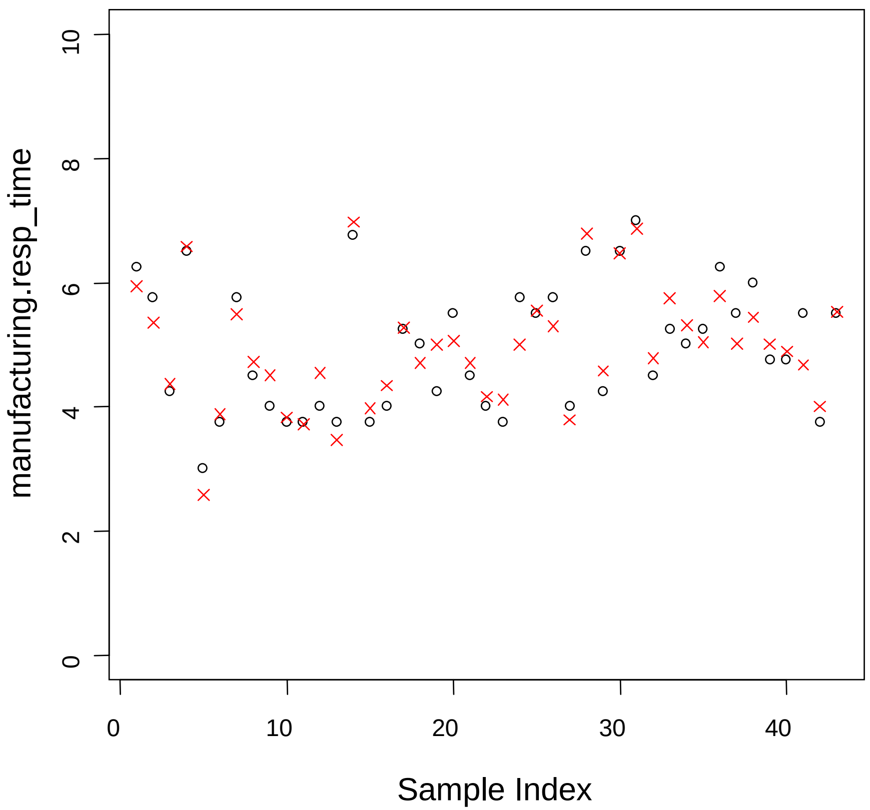
$\therefore$ 4 -> 5 nonlinear function approximation

Model Construction

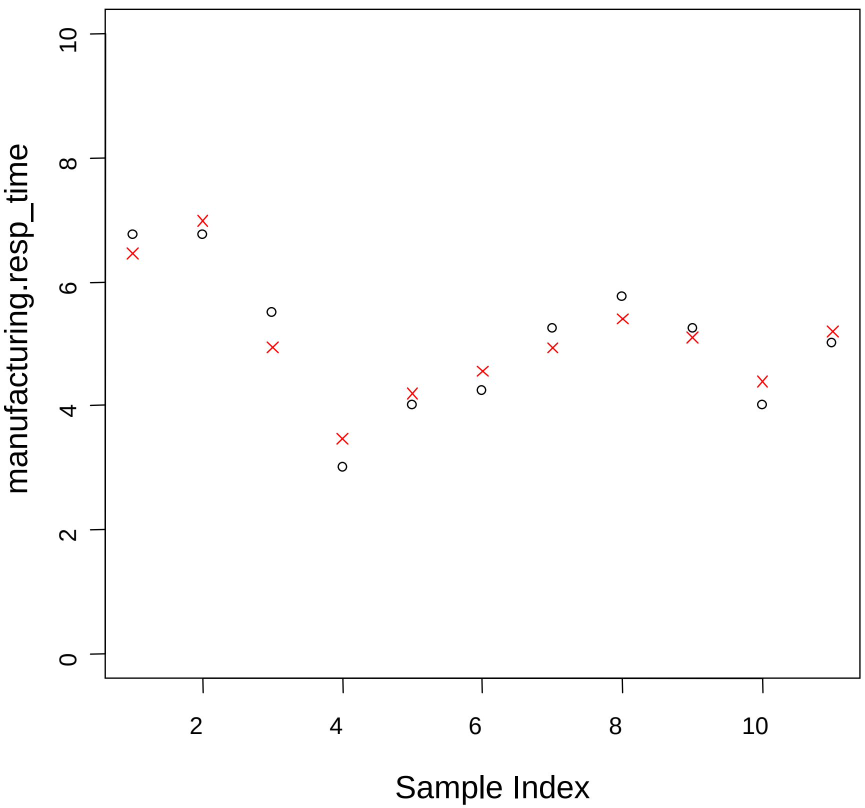
- Collected 54 data samples with varying configurations
- Train the neural network with R statistical analysis toolkit
  - Single hidden layer
  - 100 hidden nodes
  - Maximum iteration = 120
- Performed 5-fold cross validation over the model

Model Validation: Manufacturing Response Time

o : actual value
x : predicted value



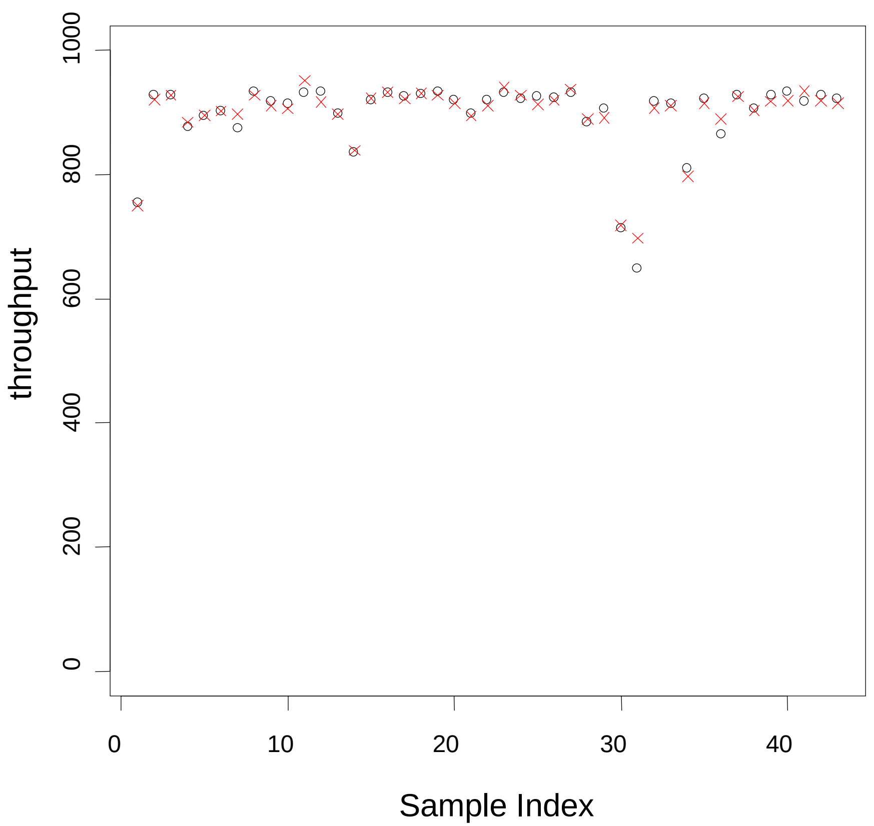
Prediction for training set



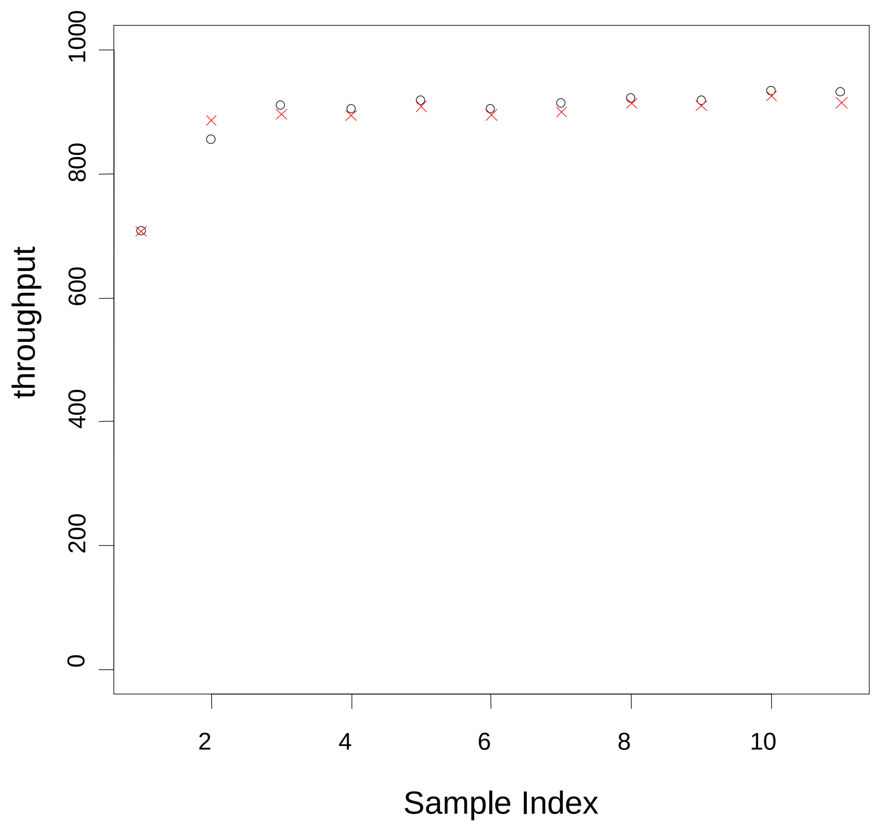
Prediction for validation set

Model Validation: Throughput

o : actual value
x : predicted value



Prediction for training set



Prediction for validation set

Model Accuracy

- Average prediction error for validation set

Trial	Manufacturing Response Time	Dealer Purchase Response Time	Dealer Manage Response Time	Dealer Browse Autos Response Time	Effective Transactions per second
1	3.30%	10.10%	5.70%	9.50%	0.10%
2	1.50%	7.30%	2.70%	4.20%	0.30%
3	4.50%	8.90%	3.30%	5.00%	0.20%
4	4.00%	12.60%	12.60%	11.30%	0.10%
5	1.40%	11.30%	10.70%	6.40%	0.20%
Average	**3.00%**	**10.00%**	**7.00%**	**7.30%**	**0.20%**

- Harmonic mean of model accuracy = 95%

Model Application

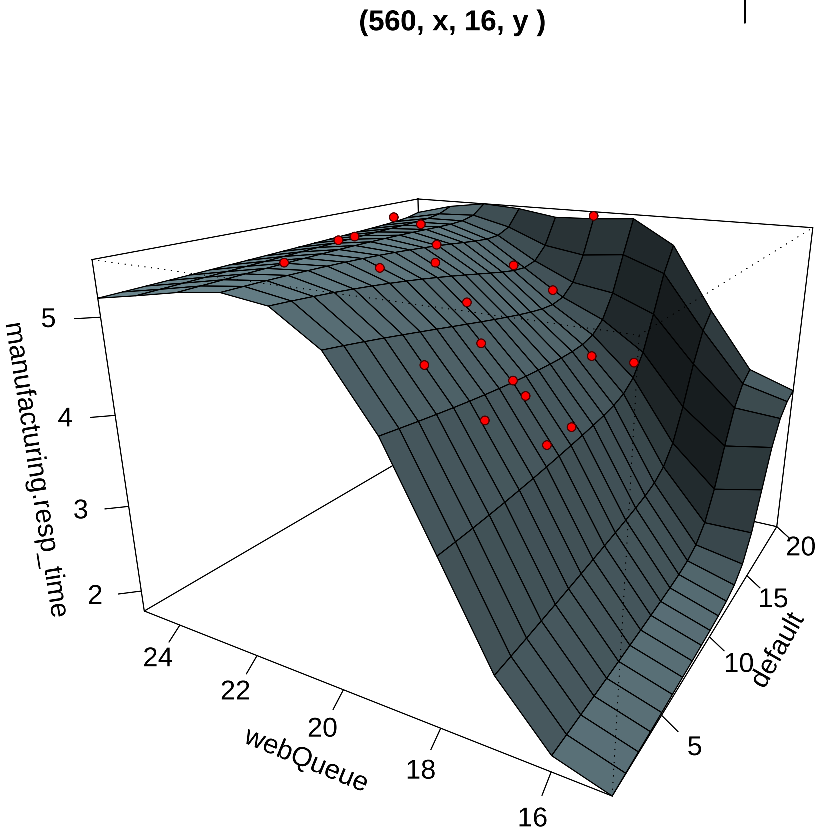
- Now we have an accurate and valid model

- Utilize this model to further improve the understandings in the workload
  - Project the model to 3D by fixing 2 out of 4 configuration parameters

- 3 typical behaviors appeared repetitively
  - Case of Parallel Slopes
  - Case of Valleys
  - Case of Hills

Case of Parallel Slopes

- Parallel Slopes
  - Injection rate and manufacturing queue fixed at (560, 16)
  - Z axis: manufacturing response time
  - X, Y axis: web queue and default queue value

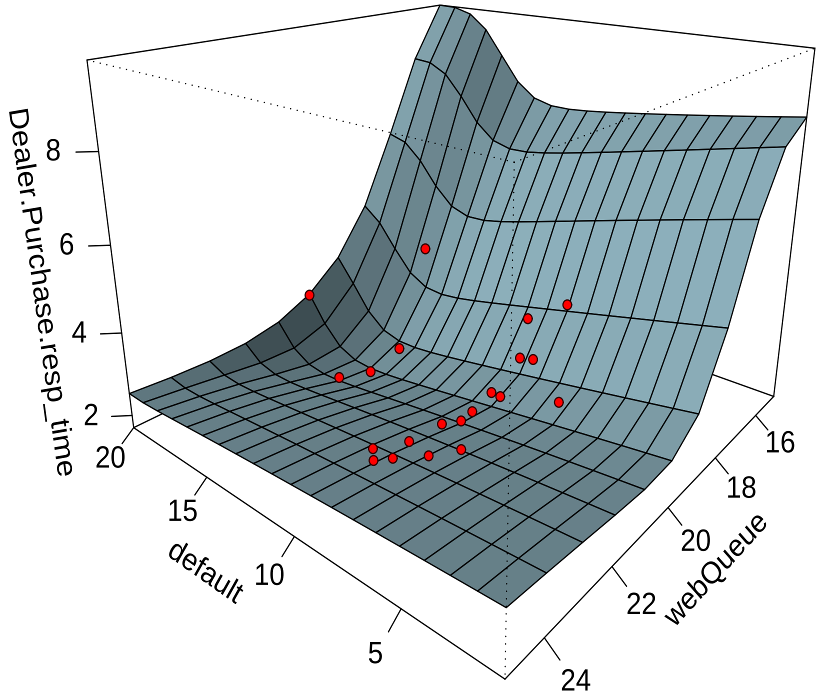


Tuning default queue value has less effect on response time once web queue value is fixed

Case of Valleys

(560, x, 16, y)

- Valleys
  - Injection rate and manufacturing queue fixed at (560, 16)
  - Z axis: dealer purchase response time
  - X, Y axis: default queue and web queue value
  - Valleys formed at (default, webQueue) = (15, 18)

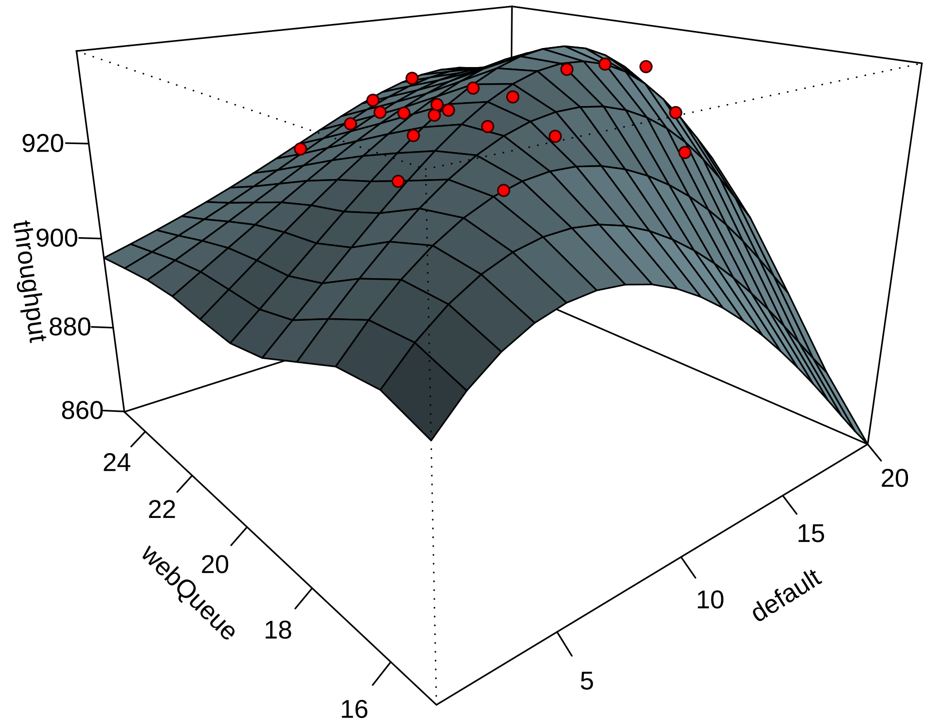


Default queue value and web queue value should be adjusted in a coherent way to stay in the 'valley'

Case of Hills

- Hills
  - Injection rate and manufacturing queue fixed at (560, 16)
  - Z axis: throughput
  - X, Y axis: web queue and default queue value

(560, x, 16, y)



Default queue value and web queue value should be adjusted in a coherent way to stay on the 'hill'

Conclusion

- Devised a methodology that incorporates neural network to construct and validate a nonlinear behavior model
- Neural networks are an excellent tool to construct a nonlinear workload behavior model
- Significant insights can be gained by analyzing these constructed models

Questions?

- Georgia Tech MARS lab

  http://arch.ece.gatech.edu/

Additional Thoughts

- Neural network models perform interpolation among samples
- Cannot be used for extrapolation
  - Cannot predict the performance for the configuration that is far apart from the training data
- Known limitation of MLP