Machine Learning Applications for Data Center Optimization

Jim Gao, Google

Abstract

The modern data center (DC) is a complex interaction of multiple mechanical, electrical and controls systems. The sheer number of possible operating configurations and nonlinear interdependencies make it difficult to understand and optimize energy efficiency. We develop a neural network framework that learns from actual operations data to model plant performance and predict PUE within a range of 0.004 +/- 0.005 (mean absolute error +/- 1 standard deviation), or 0.4% error for a PUE of 1.1. The model has been extensively tested and validated at Google DCs. The results demonstrate that machine learning is an effective way of leveraging existing sensor data to model DC performance and improve energy efficiency.

1. Introduction

The rapid adoption of Internet-enabled devices, coupled with the shift from consumer-side computing to SaaS and cloud-based systems, is accelerating the growth of large-scale data centers (DCs). Driven by significant improvements in hardware affordability and the exponential growth of Big Data, the modern Internet company encompasses a wide range of characteristics including personalized user experiences and minimal downtime. Meanwhile, popular hosting services such as Google Cloud Platform and Amazon Web Services have dramatically reduced upfront capital and operating costs, allowing companies with smaller IT resources to scale quickly and efficiently across millions of users. These trends have resulted in the rise of large-scale DCs and their corresponding operational challenges.

One of the most complex challenges is power management. Growing energy costs and environmental responsibility have placed the DC industry under increasing pressure to improve its operational efficiency. According to Koomey, DCs comprised 1.3% of the global energy usage in 2010 [1]. At this scale, even relatively modest efficiency improvements yield significant cost savings and avert millions of tons of carbon emissions.

While it is well known that Google and other major Internet companies have made significant strides towards improving their DC efficiency, the overall pace of PUE reduction has slowed given diminishing returns and the limitations of existing cooling technology [2]. Furthermore, best practice techniques such as hot air containment, water side economization, and extensive monitoring are now commonplace in large-scale DCs [3]. Figure 1 demonstrates Google's historical PUE performance from an annualized fleet-wide PUE of 1.21 in 2008 to 1.12 in 2013, due to implementation of best practices and natural progression down the learning curve [4]. Note the asymptotic decline of the trailing twelve-month (TTM) PUE graph.

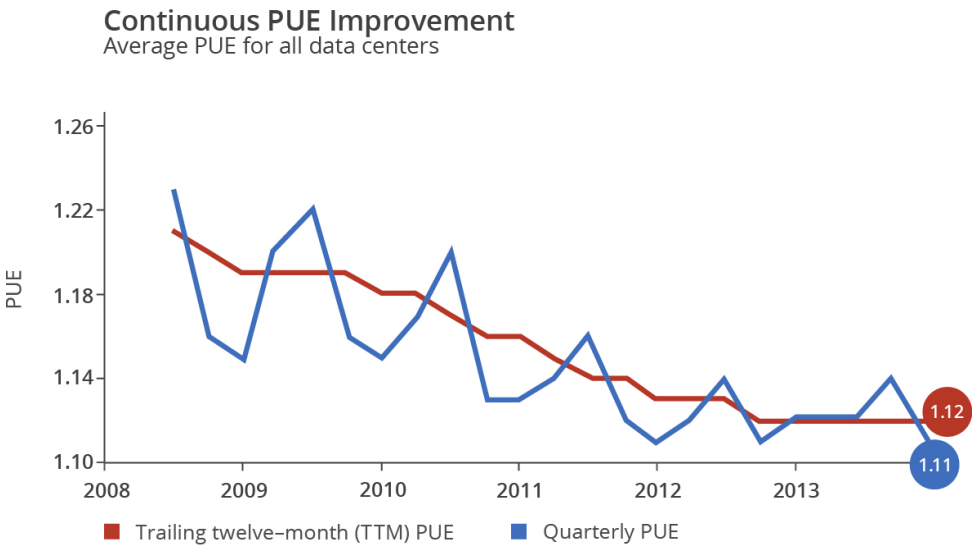


Fig 1. Historical PUE values at Google.

The application of machine learning algorithms to existing monitoring data provides an opportunity to significantly improve DC operating efficiency. A typical large-scale DC generates millions of data points across thousands of sensors every day, yet this data is rarely used for applications other than monitoring purposes. Advances in processing power and monitoring capabilities create a large opportunity for machine learning to guide best practice and improve DC efficiency. The objective of this paper is to demonstrate a data-driven approach for optimizing DC performance in the sub-1.10 PUE era.

2. Methodology

2.1 General Background

Machine learning is well-suited for the DC environment given the complexity of plant operations and the abundance of existing monitoring data. The modern large-scale DC has a wide variety of mechanical and electrical equipment, along with their associated setpoints and control schemes. The interactions between these systems and various feedback loops make it difficult to accurately predict DC efficiency using traditional engineering formulas.

For example, a simple change to the cold aisle temperature setpoint will produce load variations in the cooling infrastructure (chillers, cooling towers, heat exchangers, pumps, etc.), which in turn cause nonlinear changes in equipment efficiency. Ambient weather conditions and equipment controls will also impact the resulting DC efficiency. Using standard formulas for predictive modeling often produces large errors because they fail to capture such complex interdependencies.

Furthermore, the sheer number of possible equipment combinations and their setpoint values makes it difficult to determine where the optimal efficiency lies. In a live DC, it is possible to meet the target setpoints through many possible combinations of hardware (mechanical and electrical equipment) and software (control strategies and setpoints). Testing each and every feature combination to maximize efficiency would be unfeasible given time constraints, frequent fluctuations in the IT load and weather conditions, as well as the need to maintain a stable DC environment.

To address these challenges, a neural network is selected as the mathematical framework for training DC energy efficiency models. Neural networks are a class of machine learning algorithms that mimic cognitive behavior via interactions between artificial neurons [6]. They are advantageous for modeling intricate systems because neural networks do not require the user to predefine the feature interactions in the model, which assumes relationships within the data. Instead, the neural network searches for patterns and interactions between features to automatically generate a best-fit model. Common applications for this branch of machine learning include speech recognition, image processing, and autonomous software agents. As with most learning systems, the model accuracy improves over time as new training data is acquired.

2.2 Model Implementation

A generic three-layered neural network is illustrated in Figure 2. In this study, the input matrix x is an $(m\,x\,n)$ array where m is the number of training examples and n is the number of features (DC input variables) including the IT load, weather conditions, number of chillers and cooling towers running, equipment setpoints, etc. The input matrix x is then multiplied by the model parameters matrix θ^{1} to produce the hidden state matrix a [6]. In practice, a acts as an intermediary state that interacts with the second parameters matrix θ^{2} to calculate the output $h_{\theta}(x)$ [6]. The size and number of hidden layers can be varied to model systems of varying complexity.

Note that $h_{\theta}(x)$ is the output variable of interest and can represent a range of metrics that we wish to optimize. PUE is selected here to represent DC operational efficiency, with recognition that the metric is a ratio and not indicative of total facility-level energy consumption. Other examples include using server utilization data to maximize machine productivity, or equipment failure data to understand how the DC environment impacts reliability. The neural network will search for relationships between data features to generate a mathematical model that describes $h_{\theta}(x)$ as a function of the inputs. Understanding the underlying mathematical behavior of $h_{\theta}(x)$ allows us to control and optimize it.

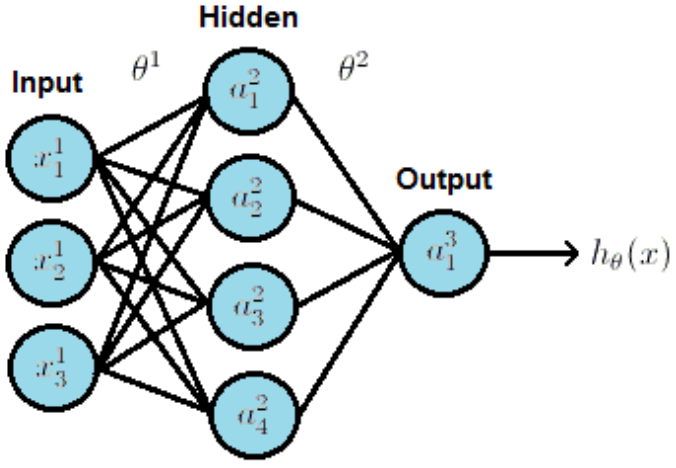


Fig. 2 Three-layer neural network.

Although linear independence between features is not required, doing so can significantly reduce the model training time, as well as the chances of overfitting [8]. Additionally, linear independence can simplify model

complexity by limiting the number of inputs to only those features fundamental to DC performance. For example, the DC cold aisle temperature may not be a desirable input for predicting PUE because it is a consequence of variables more fundamental to DC control, such as the cooling tower leaving condenser water temperature and chilled water injection setpoints.

The process of training a neural network model can be broken down into four steps, each of which are covered in greater detail below: (1) Randomly initialize the model parameters θ, (2) Implement the forward propagation algorithm, (3) Compute the cost function $J(\theta)$, (4) Implement the back propagation algorithm and (5) Repeat steps 2 - 4 until convergence or the desired number of iterations [7].

2.2.1 Random Initialization

Random initialization is the process of randomly assigning θ values between [-1, 1] before starting model training. To understand why this is necessary, consider the scenario in which all model parameters are initialized at 0. The inputs into each successive layer in the neural network would then be identical, since they are multiplied by θ. Furthermore, since the error is propagated backwards from the output layer through the hidden layers, any changes to the model parameters would also be identical [7]. We therefore randomly initialize θ with values between [-1, 1] to avoid the formation of unstable equilibriums [7].

2.2.2 Forward Propagation

Forward propagation refers to the calculation of successive layers, since the value of each layer depends upon the model parameters and layers before it. The model output $h_\theta(x)$ is computed through the forward propagation algorithm, where a_j^l represents the activation of node j in layer l, and θ^l represents the matrix of weights (model parameters) mapping layer l to layer $l+1$.

$$
\begin{aligned}
a_1^2 &= g(\theta_{10}^1 x_0^1 + \theta_{11}^1 x_1^1 + \theta_{12}^1 x_2^1 + \theta_{13}^1 x_3^1) \\
a_2^2 &= g(\theta_{20}^1 x_0^1 + \theta_{21}^1 x_1^1 + \theta_{22}^1 x_2^1 + \theta_{23}^1 x_3^1) \\
a_3^2 &= g(\theta_{30}^1 x_0^1 + \theta_{31}^1 x_1^1 + \theta_{32}^1 x_2^1 + \theta_{33}^1 x_3^1) \\
a_4^2 &= g(\theta_{40}^1 x_0^1 + \theta_{41}^1 x_1^1 + \theta_{42}^1 x_2^1 + \theta_{43}^1 x_3^1) \\
h_\theta(x) = a_1^3 &= g(\theta_{10}^2 a_0^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2 + \theta_{14}^2 a_4^2)
\end{aligned}
$$

Bias units (nodes with a value of 1) are appended to each non-output layer to introduce a numeric offset within each layer [6]. In the equations above, θ_{10}^1 represents the weight between the appended bias unit x_0^1 and the hidden layer element a_1^2.

The purpose of the activation function $g(z)$ is to mimic biological neuron firing within a network by mapping the nodal input values to an output within the range (0, 1). It is given by the sigmoidal logistic function $g(z) = 1/(1+e^{-z})$ [6]. Note that the equations above can be expressed more compactly in matrix form as:

$$
\begin{aligned}
a^2 &= g(\theta^1 x) \\
h_\theta(x) = a^3 &= g(\theta^2 a^2)
\end{aligned}
$$

2.2.3 Cost Function

The cost function $J(\theta)$ serves as the quantity to be reduced with each iteration during model training. It is typically expressed as the square of the error between the predicted and actual outputs. For linear regression problems, the cost function can be expressed as:

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2 + \lambda \sum_{i=1}^{L-1} \sum_{j=1}^{n} \theta_{ij}^2]$$

where $h_\theta(x)$ is the predicted output, y is the actual data corresponding output variable of interest, m is the number of training examples per feature, L is the number of layers, and n is the number of nodes [7]. The regularization parameter λ controls the tradeoff between model accuracy and overfitting [5]. In this example, $h_\theta(x)$ is the calculated PUE through the neural network and y is the actual PUE data.

2.2.4 Back Propagation

After computing the cost function $J(\theta)$, the error term δ is propagated backwards through each layer to refine the values of θ. The error for the output layer is defined as the difference between the calculated output $h_\theta(x)$ and the actual output y. For the three-layered network in Fig. 2, the errors associated with the output and hidden layers are calculated as:

$$\delta^3 = a^3 - y$$
$$\delta^2 = (\theta^2)^T \delta^3 .* g'(z^2) = (\theta^2)^T \delta^3 .* [a^2 .* (1 - a^2)]$$

where $g'(z)$ represents the derivative of the activation function [7]. Note that $g'(z)$ simplifies to the expression $a .* (1 - a)$. There is no error term associated with the first layer because it is the input layer. The theta gradient vector D^l, computed for each layer l, is then calculated as:

$$\Delta^l = \Delta^l + \delta^{(l+1)} a^l$$
$$D^l = \frac{1}{m}(\Delta^l + \lambda\theta^l) \quad \text{if } j \neq 0$$
$$D^l = \frac{1}{m}\Delta^l \quad \text{if } j = 0$$

where Δ^l is initialized as a vector of zeros [7]. D^l is added to θ^l to update the each layer's model parameters before repeating steps 2 - 4 in the next iteration. As with all machine learning algorithms, it will take a varying number of iterations (typically in the hundreds or thousands) until convergence, as approximated by sufficiently small reductions in the cost function $J(\theta)$.

2.3 Implementation

The neural network utilizes 5 hidden layers, 50 nodes per hidden layer and 0.001 as the regularization parameter. The training dataset contains 19 normalized input variables and one normalized output variable (the DC PUE), each spanning 184,435 time samples at 5 minute resolution (approximately 2 years of operational data). 70% of the dataset is used for training with the remaining 30% used for cross-validation and testing. The chronological order of the dataset is randomly shuffled before splitting to avoid biasing the training and testing sets on newer or older data.

Data normalization, also known as feature scaling, is recommended due to the wide range of raw feature values. The values of a feature vector z are mapped to the range [-1, 1] by:

$$z_{norm} = \frac{z - MEAN(z)}{MAX(z) - MIN(z)}$$

The neural network features are listed as follows:
1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed [mph]
19. Outdoor wind direction [deg]

Note that many of the inputs representing totals and averages are actually meta-variables derived from individual sensor data. Data pre-processing such as file I/O, data filtration and calculating meta-variables was conducted using Python2.7 in conjunction with the Scipy 0.12.0 and Numpy 1.7.0 modules. Matlab R2010a was used for model training and post-processing. Open source alternatives offering similar functionality to Matlab R2010a include Octave as well as the Scipy/Numpy modules in Python.

3. Results and Discussion

Having an accurate and robust PUE model offers many benefits for DC operators and owners. For example, the comparison of actual vs. predicted DC performance for any given set of conditions can be used for automatic performance alerting, real-time plant efficiency targets and troubleshooting.

A robust efficiency model also enables DC operators to evaluate PUE sensitivity to DC operational parameters. For example, an internal analysis of PUE versus cold aisle temperature (CAT) conducted at a Google DC suggested a theoretical 0.005 reduction in PUE by increasing the cooling tower LWT and chilled water injection pump setpoints by 3F (see Section 3.3). This simulated PUE reduction was subsequently verified with experimental test results after normalizing for server IT load and wet bulb temperature. Such sensitivity analyses drive significant cost and carbon savings by locating and estimating the magnitude of opportunities for further PUE reductions.

Finally, a comprehensive DC efficiency model enables operators to simulate the DC operating configurations without making physical changes. Currently, it's very difficult for an operator to predict the effect of a plant configuration change on PUE prior to enacting the changes. This is due to the complexity of modern DCs,

and the interactions between multiple control systems. A machine learning approach leverages the plethora of existing sensor data to develop a mathematical model that understands the relationships between operational parameters and the holistic energy efficiency. This type of simulation allows operators to virtualize the DC for the purpose of identifying optimal plant configurations while reducing the uncertainty surrounding plant changes.

3.1 Predictive Accuracy

Figure 3 depicts a snapshot of predicted vs actual PUE values at one of Google's DCs over one month during the summer.

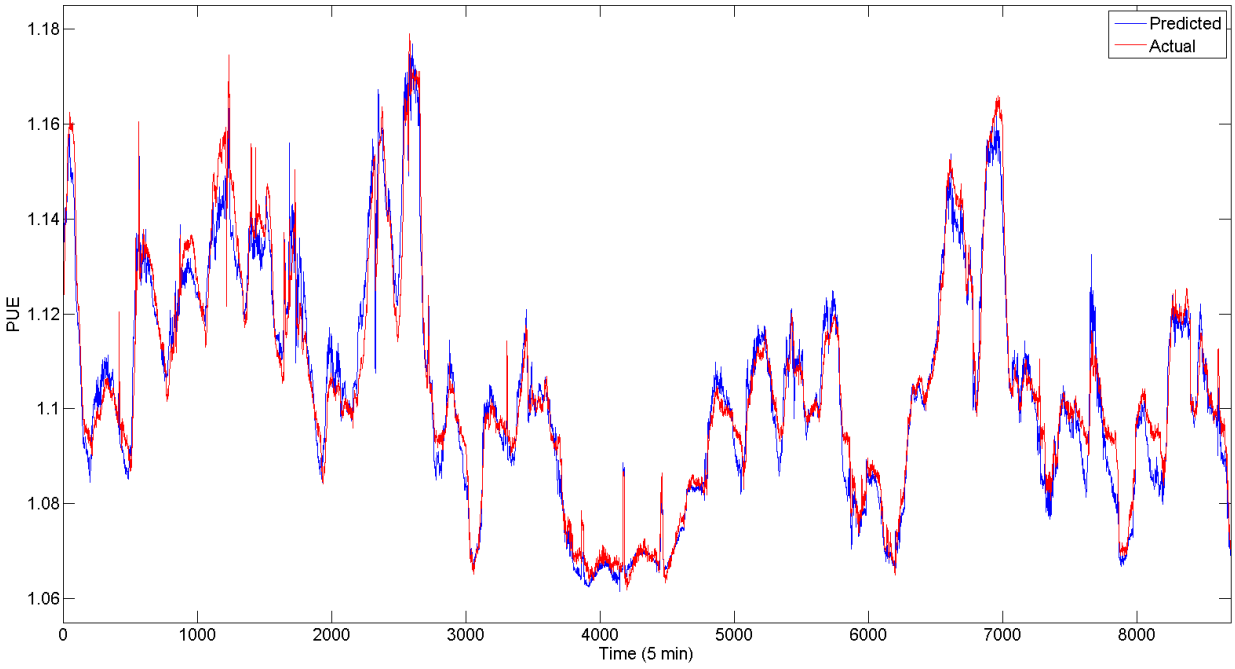
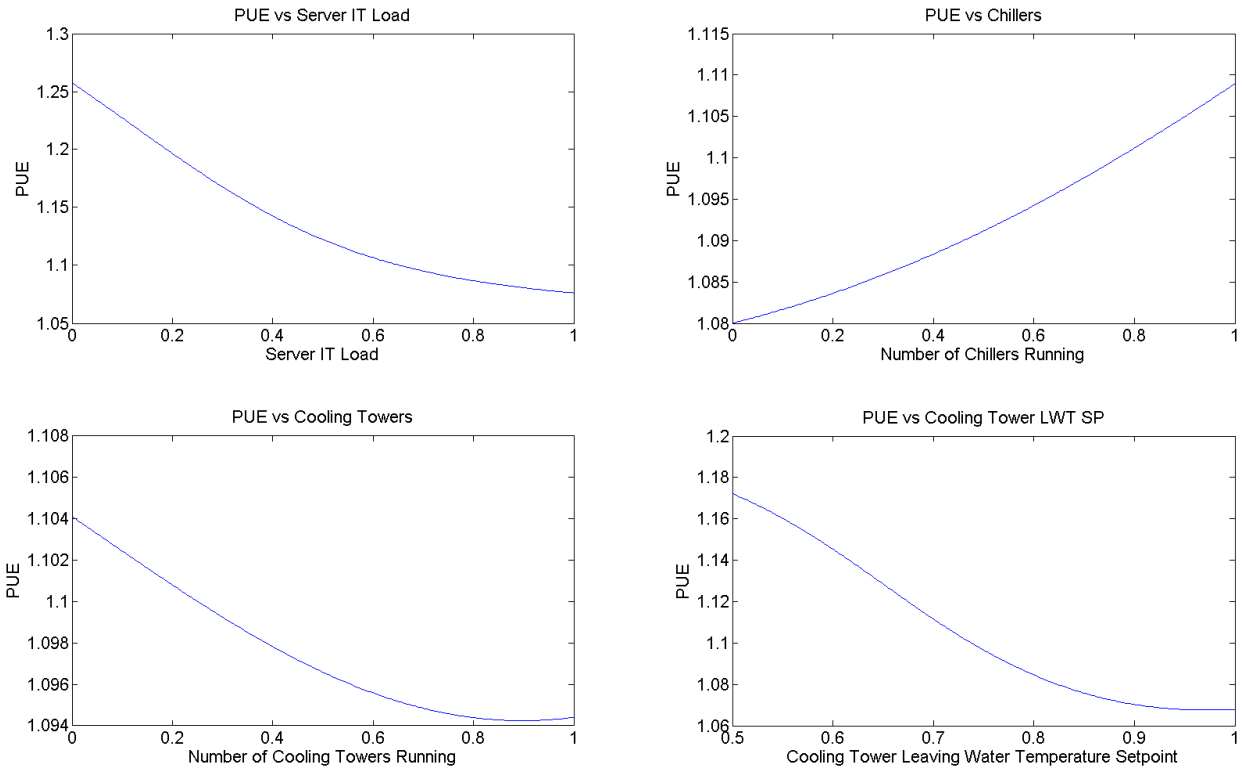


Fig. 3 Predicted vs actual PUE values at a major DC.

The neural network detailed in this paper achieved a mean absolute error of 0.004 and standard deviation of 0.005 on the test dataset. Note that the model error generally increases for PUE values greater than 1.14 due to the scarcity of training data corresponding to those values. The model accuracy for those PUE ranges is expected to increase over time as Google collects additional data on its DC operations.

3.2 Sensitivity Analysis

The following graphs reveal the impact of individual operating parameters on the DC PUE. We isolate for the effects of specific variables by linearly varying one input at a time while holding all others constant. Such sensitivity analyses are used to evaluate the impact of setpoint changes and identify optimal setpoints. All test results have been verified empirically.

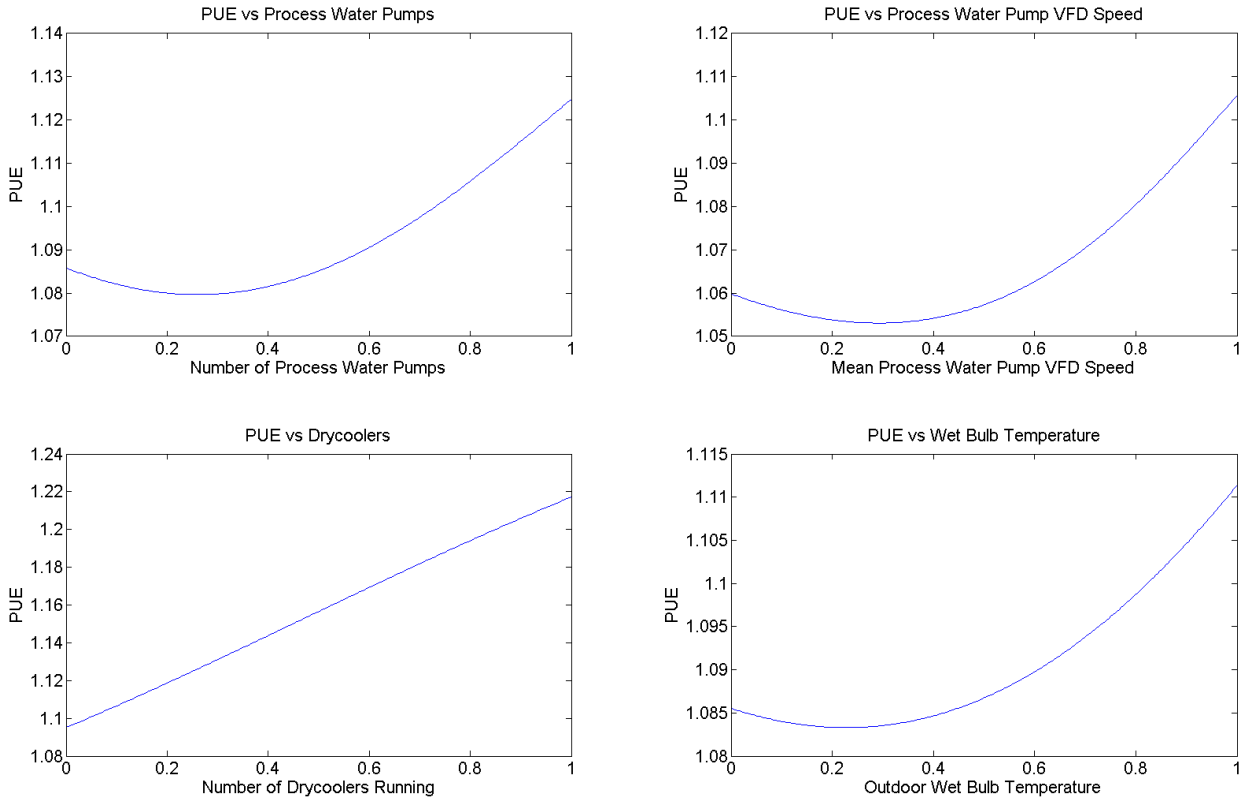


Figs. 4a-d PUE vs Server IT load, Chillers, Cooling Towers and Cooling Tower LWT setpoint (normalized).

Fig. 4a demonstrates the relationship between PUE and Server IT load, with large efficiency gains occurring in the 0 - 70% max load range and an asymptotic decline beyond 70%. This is field-verified through Google's historical experience in which PUE decreases rapidly initially with increasing IT load due to economies of scale and more efficient utilization of the cooling plant. The PUE vs IT curve gradually levels off as the cooling plant nears its maximum efficiency and operating capacity.

Fig. 4b shows the relationship between PUE and the number of operational chillers. As expected, there are significant PUE increases associated with bringing more chillers online. The relationship between PUE and the number of chillers running is nonlinear because chiller efficiency decreases exponentially with reduced load.

Figs. 4c and 4d illustrate the relationship between PUE and the number of cooling towers running, as well as the mean cooling tower leaving water temperature (LWT) setpoint, respectively. From the Fan Affinity Laws, we expect the PUE to decrease with as additional cooling towers are brought online due to the inverse cubic relationship between fan speed and power consumption. Splitting the same cooling load across more towers requires a lower mean fan speed per tower. Likewise, in Fig. 4d, less power is consumed at the plant level as the mean cooling tower LWT setpoint increases. This is due to the linear reduction in required fan speed and the corresponding cubic reduction in fan power. Fig. 4c suggests that a shared condenser water piping design, whereby cooling load is split between multiple towers, is more efficient at the plant level than individual pipes supplying modular towers.



Figs 5a-d PUE vs PWP, mean PWP VFD speed, drycoolers and wet bulb temperature (normalized).

Figs. 5a and 5b demonstrate the relationship between DC PUE and the number of process water pumps (PWP) running, as well as the mean process water pump speed, respectively. For a fixed pump speed, increasing the number of PWPs increases the total cooling overhead and thus the PUE. Likewise, increasing the mean PWP speed while holding the number of operational PWPs fixed results in a cubic increase in pump overhead energy.

Fig. 5c illustrates the effect of running drycoolers on PUE. Drycoolers are used infrequently during winter months when ambient weather conditions pose significant risk of cooling tower icing. Since drycoolers utilize an intermediary, closed-circuit fluid to exchange heat between the process water and outdoor air, they typically exhibit lower efficiencies than traditional cross-flow or counter-flow cooling towers. This is reflected in the PUE graph, which shows large, linear increases in cooling overhead with additional drycoolers.

Fig. 5d shows the effect of increasing wet bulb temperature on PUE. The shape of the curve matches well against Google's historical data - higher wet bulb temperatures limit cooling tower range, requiring higher fan speeds and additional mechanical cooling. The slight PUE increase at low wet bulb temperatures is due to drycooler usage.

Fig. 6 demonstrates the relationship between PUE and outside air enthalpy, or total energy content of the ambient air. As the air enthalpy increases, the number of cooling towers, supplemental chillers, and associated loading rises as well, producing a nonlinear effect on the DC overhead. Note that enthalpy is a more comprehensive measure of outdoor weather conditions than the wet bulb temperature alone since it includes the moisture content and specific heat of ambient air.

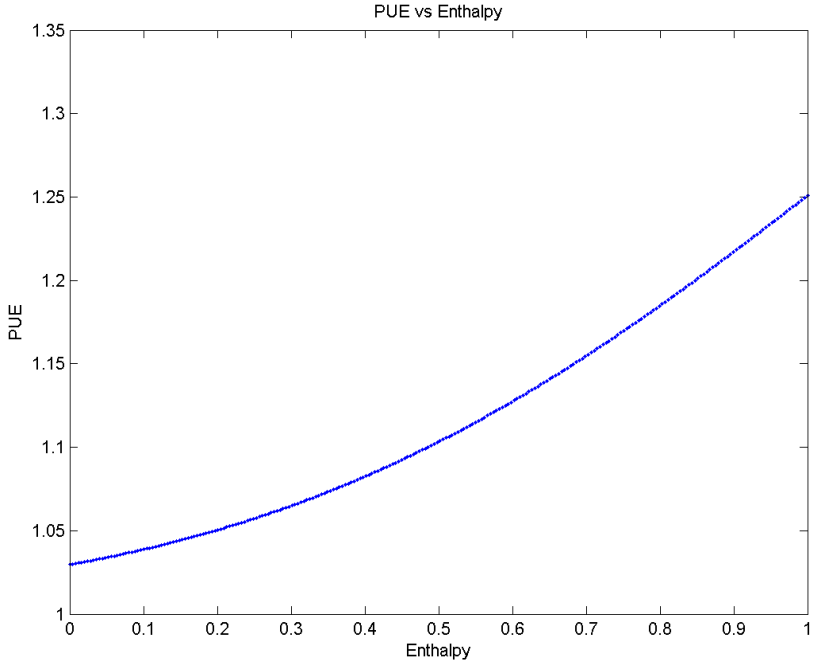


Fig. 6 PUE vs outside air enthalpy.

The sensitivity analyses show that outdoor air enthalpy has the largest impact on DC PUE, followed closely by the IT load. The cooling tower LWT setpoint and number of operational drycoolers also significantly impact the facility PUE.

3.3 Application Examples

The following case studies represent concrete applications of predictive modeling to DC optimization. In the interest of brevity, only three examples are presented.

Example 1: Simulating Process Water Supply Temperature Increases

In this study, we increased the process water supply (PWS) temperature to the server floor by 3F. This was accomplished by raising the cooling tower LWT and the chilled water injection pump temperature setpoints. Fig. 4d predicts a ~0.005 PUE decrease as a result of the cooling tower LWT setpoint increase.

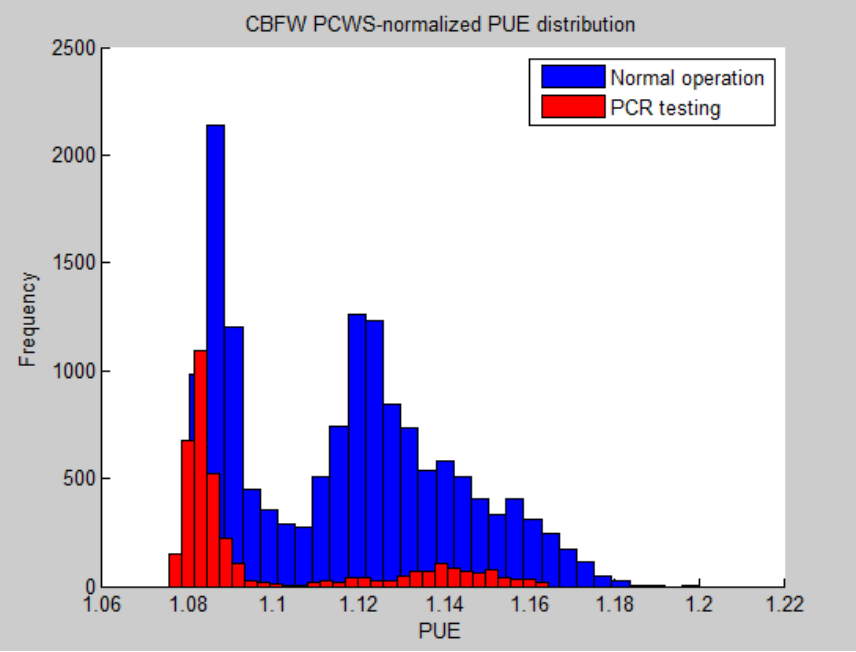


Fig. 7 PUE distributions corresponding to higher PWS temp (red) vs lower PWS temp (blue).

Fig. 7 shows the PUE distribution for 3 weeks of operation at the higher PWS temperature (shown in red), superimposed over 1.5 years of operation at the lower PWS temp (shown in blue). The datasets are filtered for similar server load and wet bulb conditions. Both distributions are clearly bimodal, with the left sides corresponding to cooling tower operation only and the right sides corresponding to cooling tower and chiller operation during summertime conditions. As predicted, there is a ~0.005 difference between the mean PUE values of the left sides of the distributions. This PUE difference is not mirrored on the right sides of the distributions because the chillers do not come online until the cooling towers are already at max capacity.

Example 2: Catching Erroneous Meter Readings

In Q2-2011, Google announced that it would include natural gas as part of ongoing efforts to calculate PUE in a holistic and transparent manner [9]. This required installing automated natural gas meters at each of Google's DCs. However, local variations in the type of gas meter used caused confusion regarding erroneous measurement units. For example, some meters reported 1 pulse per 1000 scf of natural gas, whereas others reported a 1:1 or 1:100 ratio. The local DC operations teams detected the anomalies when the real-time, actual PUE values exceeded the predicted PUE values by 0.02 - 0.1 during periods of natural gas usage.

Example 3: DC Plant Configuration Optimization

In this study, a planned upgrade to the electrical infrastructure required rerouting ~40% of the server traffic for several days for worker safety. This required corresponding changes in the operational lineup to meet the reduced IT load while maintaining DC efficiency. Through a combination of PUE simulations and local expertise, the team selected a new set of operational parameters that reduced the PUE by ~0.02 compared to the previous configuration.

4. Limitations

Machine learning applications are limited by the quality and quantity of the data inputs. As such, it is important to have a full spectrum of DC operational conditions to accurately train the mathematical model. The model accuracy may decrease for conditions where there is less data. As with all empirical curve-fitting, the same predictive accuracy may be achieved for multiple model parameters θ. It is up to the analyst and DC operator to apply reasonable discretion when evaluating model predictions.

5. Conclusion

Accelerating growth in DC complexity and scale is making energy efficiency optimization increasingly important yet difficult to achieve. Using the machine learning framework developed in this paper, we are able to predict DC PUE within 0.004 +/- 0.005, approximately 0.4% error for a PUE of 1.1. Actual testing on Google DCs indicate that machine learning is an effective method of using existing sensor data to model DC energy efficiency, and can yield significant cost savings. Model applications include DC simulation to evaluate new plant configurations, assessing energy efficiency performance, and identifying optimization opportunities.

Acknowledgements

I would like to thank Tal Shaked for his insights on neural network design and implementation. Alejandro Lameda Lopez and Winnie Lam have been instrumental in model deployment on live Google data centers. Finally, this project would not have been possible without the advice and technical support from Joe Kava, as well as the local data center operations teams.

References

[1] Jonathan Koomey. *Growth in Data center electricity use 2005 to 2010*. Analytics Press, Oakland, CA, 2011.

[2] Uptime Institute. *2013 Data Center Industry Survey*. 2013.

[3] *Google's Green Data Centers: Network POP Case Study*. 2011.

[4] Retrieved from https://www.google.com/about/datacenters/efficiency/internal/.

[5] Andrew Ng. "*Regularization (Week 3)*." Machine Learning. Retrieved from https://class.coursera.org/ml-2012-002. 2012. Lecture.

[6] Andrew Ng. "*Neural Networks: Representation (Week 4)*." Machine Learning. Retrieved from https://class.coursera.org/ml-2012-002. 2012. Lecture.

[7] Andrew Ng. "*Neural Networks: Learning (Week 5)*." Machine Learning. Retrieved from https://class.coursera.org/ml-2012-002. 2012. Lecture.

[8] Andrew Ng. "*Advice for Applying Machine Learning (Week 6)*." Machine Learning. Retrieved from https://class.coursera.org/ml-2012-002. 2012. Lecture.

[9] Retrieved from http://www.google.com/about/datacenters/efficiency/internal/#tab0=10.