Synthesis of Reinforcement Learning, Neural Networks, and PI Control Applied to a Simulated Heating Coil

Charles W. Anderson[1], Douglas C. Hittle[2], Alon D. Katz[2], and R. Matt Kretchmar[1]

[1]Department of Computer Science
Colorado State University
Fort Collins, CO 80523
{anderson,kretchma}@cs.colostate.edu

[2]Department of Mechanical Engineering
Colorado State University
Fort Collins, CO 80523
{hittle,alon}@lamar.colostate.edu

Abstract

An accurate simulation of a heating coil is used to compare the performance of a proportional plus integral (PI) controller, a neural network trained to predict the steady-state output of the PI controller, a neural network trained to minimize the n-step ahead error between the coil output and the set point, and a reinforcement learning agent trained to minimize the sum of the squared error over time. Although the PI controller works very well for this task, the neural networks produce improved performance. The reinforcement learning agent, when combined with a PI controller, learned to augment the PI control output for a small number of states for which control can be improved.

Keywords: neural networks, reinforcement learning, PI control, HVAC

1 Introduction

Typical methods for designing fixed feedback controllers results in sub-optimal control performance. In many situations, the degree of uncertainty in the model of the system being controlled limits the utility of optimal control design. Building energy systems are particularly troublesome since the process gain is highly variable, depending on the load on components such as heating and cooling coils and on inlet conditions such as air temperature and air volume flow rate. Some of these issues have been addressed by applying neural networks and other artificial intelligence techniques to the control of heating and air-conditioning systems.[4,5,8]

In this article, three approaches to improving the performance of an ordinary proportional plus integral controller are explored. One is a feed forward neural network controller designed to bring the controlled variable to its set point in n sampling intervals (where n is variable). The second is a feed forward neural network controller in parallel with a proportional feed back controller. In this case the neural network is trained to produce the steady state value of the control signal required to achieve the set point. The third uses a procedure for adding a reinforcement-learning component to an existing feedback controller in order to minimize the sum of the squared error of the control variable over time.

The procedures are empirically tested by applying them to an accurate simulation of a heating coil, a part a heating system for a building. A proportional plus integral (PI) controller is applied to the simulated heating coil and the controller's proportional and integral gains are set to values that result in the best performance under likely disturbances and changes in the set point. The coil simulation and PI controller are described in Section 2. The performance of this well-tuned PI controller in maintaining the set point is the basis of comparison for three approaches to improving the control using neural networks. Section 3 describes how a simple inverse model can be used to train a neural-network controller. Section 4 describes a neural network trained to predict the steady-state output of the PI controller and shows that when combined with a proportional controller, it performed better than the PI controller. In Section 5, an optimal control method based on reinforcement learning is presented. The reinforcement learning agent learns to augment the output of the PI controller only when it results in improved performance.

2 Heating Coil Model and PI Control

Underwood and Crawford[10] developed a model of an existing heating coil by fitting a set of second-order, nonlinear equations to measurements of air and water temperatures and flow rates obtained from the actual coil. A diagram of the model with a PI controller is shown in Figure 1. The state of the modeled system is defined by the air and water input and output temperatures, T_{ai}, T_{ao}, T_{wi}, T_{wo}, f_a, and f_w. The control signal, c, input to the model affects the water flow rate. The model is given by the following equations with constants determined by a least-square fit to data from an actual heating coil:

$$\begin{aligned}
f_w(t) &= 0.008 + 0.00703(-41.29 + 0.30932c(t-1) + -3.2681x10^{-4}c(t-1)^2 9.56x10^{-8}c(t-1)^3), \\
T_{wo}(t) &= T_{wo}(t-1) + 0.64908f_w(t-1)(T_{wi}(t-1) - T_{wo}(t-1)) + \\
&\quad (0.02319 + 0.10357f_w(t-1) + 0.02806f_a(t-1))(T_{ai}(t-1) - \frac{(T_{wi}(t-1) + T_{wo}(t-1))}{2}), \\
T_{ao}(t) &= T_{ao}(t-1) + 0.19739f_a(t-1)(T_{ai}(t-1) - T_{ao}(t-1)) + \\
&\quad (0.03184 + 0.15440f_w(t-1) + 0.04468f_a(t-1))(\frac{(T_{wi}(t-1) + T_{wo}(t-1))}{2} - \\
&\quad T_a i(t-1)) + 0.20569 * (T_{ai}(t) - T_{ai}(t-1)).
\end{aligned}$$

For the experiments reported here, the variables T_{ai}, T_{wi}, and f_a were modified by random walks to model the disturbances and changing conditions that would occur in actual heating and air conditioning systems. The bounds on the random walks were $4 \leq T_{ai} \leq 10^o$C, $73 \leq T_{wi} \leq 81^o$C, and $0.7 \leq f_a \leq 0.9$ kg/s.

A PI controller was tuned to control the simulated heating coil. The best proportional and integral gains

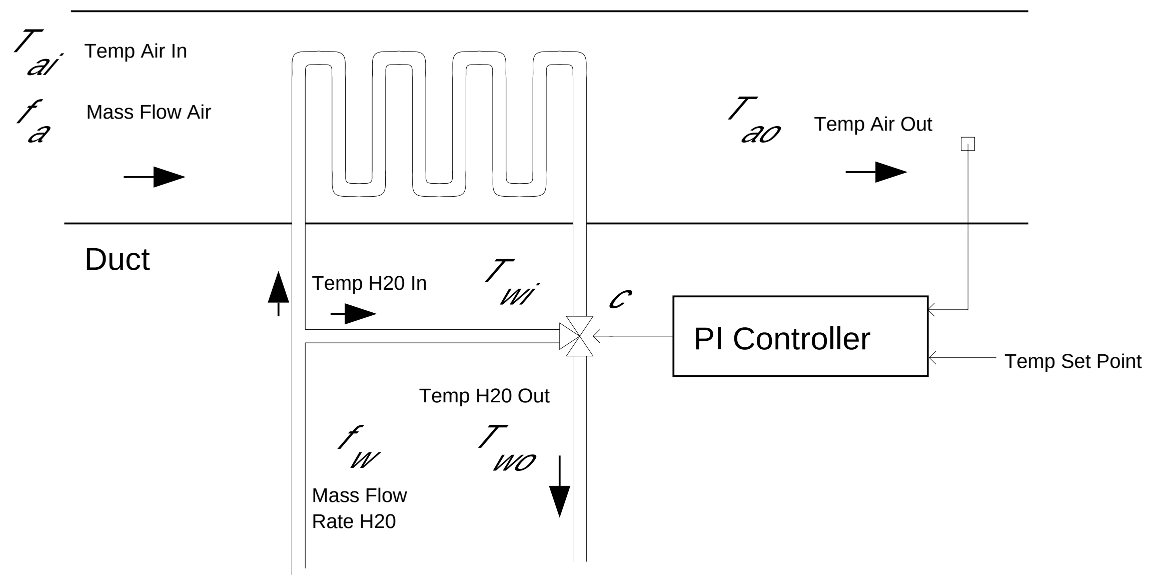


Figure 1: The simulated heating coil under control of a PI feedback controller.

were determined by measuring the sum of the absolute value of the difference between the set point and actual exiting air temperature under a variety of disturbances.

3 Training with a Simple Inverse Model

The most immediate problem that arises when designing a procedure for training a neural-network controller is that a desired output for the network is usually not available. One way to obtain such an error is to transform the error in the controlled state variable, such as a difference from a set point, into a control-signal error and use this error to train the network. An inverse model of the controlled system is needed for this transformation. In addition to an inverse model, one must consider which error in time is to be minimized. Ideally, a sum of the error over time is minimized, which is addressed in Section 5. Here the error n steps ahead is minimized.

A neural network was trained to reduce the error in the output air temperature n steps ahead. A “step” is five seconds for the simulated heating coil. This is the assumed sampling interval for a digital controller that implements any of the control schemes tried here. The network was used as an independent controller, with the output of the network being the control signal as shown in Figure 2. To train an n-step ahead network, a simple inverse model was assumed: the error of the network’s output at time step t was defined to be proportional to the difference between the output air temperature and the set point at time $t + n$. Inputs to the network were T_{ai}, T_{ao}, T_{wi}, T_{wo}, f_a, f_w, and the set point at time t. The network had these seven inputs,

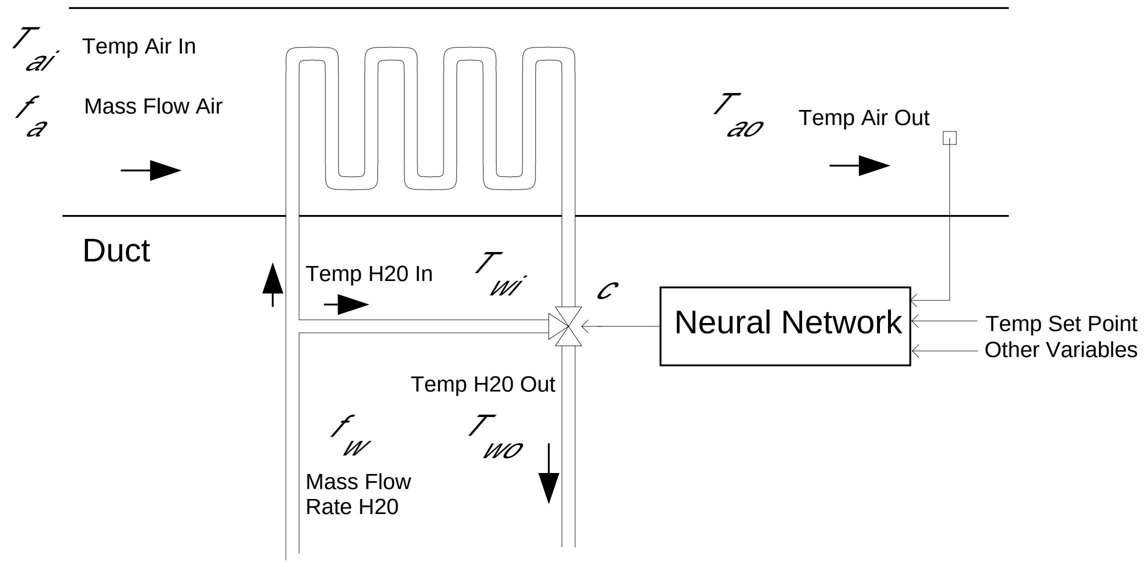


Figure 2: The heating coil under control of a neural network receiving the current and set point value of the controlled temperature plus additional measurements.

30 hidden units, and one output unit. Training data was generated by randomly changing the set point and system disturbances every 30 time steps.

Neural networks were trained to minimize the error from two to eight steps ahead. All were found to reduce the error in comparison to the PI controller by itself. For $n = 1$, 2, and 3, the RMS error between T_{ao} and its set point was about 0.6 over a 300-step test sequence. For $n = 5$ and 6, the error increased to approximately 0.7 and 0.65, respectively. Figure 3 shows a 60-step portion of the test sequence. Smaller values of n result in more overshoot of the set point. The performance of the PI controller as shown is significantly damped. Recall that the PI proportional and integral gains were chosen to minimize the RMS error over a wide variety of disturbances.

4 Prediction of PI Steady-State

The equation for the output from a PI controller is

$$O = a + k_p e_t + k_i \int e_t \, dt,$$

where k_p is the proportional gain, k_i is the integral gain, and e_t is the error between the measured and set point values of the controlled variable. A relatively slowly changing offset from the set point can be removed through integral control, which slowly adjusts the control signal to minimize the offset by adding up changes to the control signal. To maximize stability, this integration process must be slow, in general requiring many

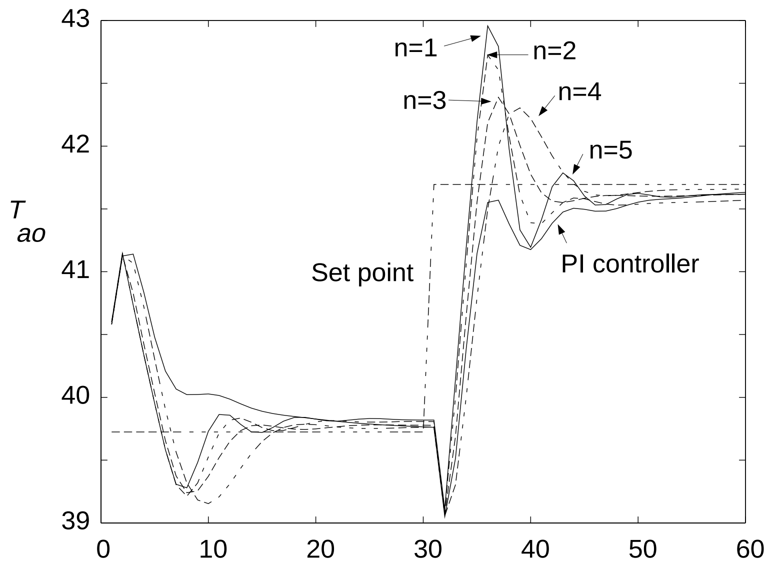


Figure 3: The set point and action output air temperature when controlled with either the PI controller, or a neural network trained to minimize the n-step ahead error.

time steps to remove the offset. If the steady-state output of an integral controller can be predicted, then the control output can be set to the predicted value immediately, without waiting for the integral term to ramp up.

We investigated this possibility by training a neural network to predict the steady-state output of the PI controller for the heating coil model. Inputs to the neural network were T_{ai}, T_{wi}, f_a, and the set point. The desired output, or target, of the network was determined by allowing the PI controller to control the heating coil simulation until steady state was reached. A data set of 10,000 input and desired output pairs was collected. Of this data, 80% was used for training, 10% for cross-validation, and 10% for testing. A two-layer network with four inputs, a variable number of hidden units, and one output unit was used.

To determine the appropriate number of hidden units, networks containing 1, 2, 3, 4 and 6 hidden units were trained for 20 epochs using traditional error back-propagation. These results are compared in Figure 4. After training for 20 epochs, the network's output for the test data differed from the target value by an average of 0.7%. Clearly, one hidden unit was sufficient; adding hidden units only increases training time. The apparent improved training performance of a two-hidden-unit network is due to a fortuitous, random selection of initial weight values.

This approach of using a neural network to duplicate the steady state output of a PI controller was proposed by Hepworth, Dexter, and Willis,[6,7] who used two inputs (discharge air temperature and inlet air tem-

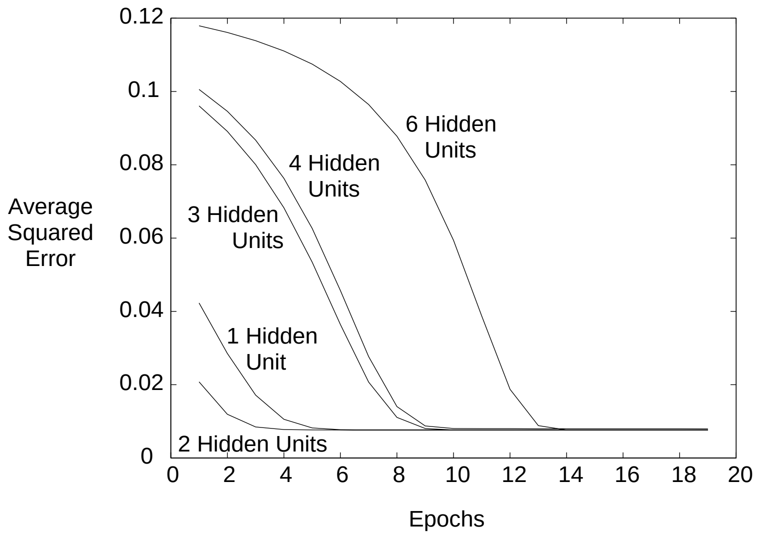


Figure 4: Average squared error versus epochs for various numbers of hidden units.

perature) to train a network of radial basis functions. In contrast, we used sigmoid neurons and took advantage of additional measurable variables such as air flow rate, inlet water temperature and water flow rate. The fact that only one hidden unit was needed to do a good job of modeling the desired steady state output, suggests that required mapping is fairly linear and that sigmoids may be a better choice than radial basis functions or other more complex functions for this application. However, we give Hepworth, Dexter, and Willis full credit for the seed of this idea.

The steady-state, neural network predictor used alone as a controller for the heating coil simulation performed poorly in comparison to the PI controller. This is to be expected, because the short term proportional component is missing. When combined with a proportional controller as shown in Figure 5, whose gain is optimized for this case, performance was significantly better than the original PI controller's performance. Figure 6 shows the behavior of the exiting air temperature when controlled with PI controller, the neural network alone, and with the combined neural network and proportional controller.

This successful simulation result in not surprising, since well-behaved physical models are modeled well by statistical methods. However, our approach is more general than an analytical solution. It can be used for systems for which a model is not well known and for situations where systems constants need to be determined by regression.

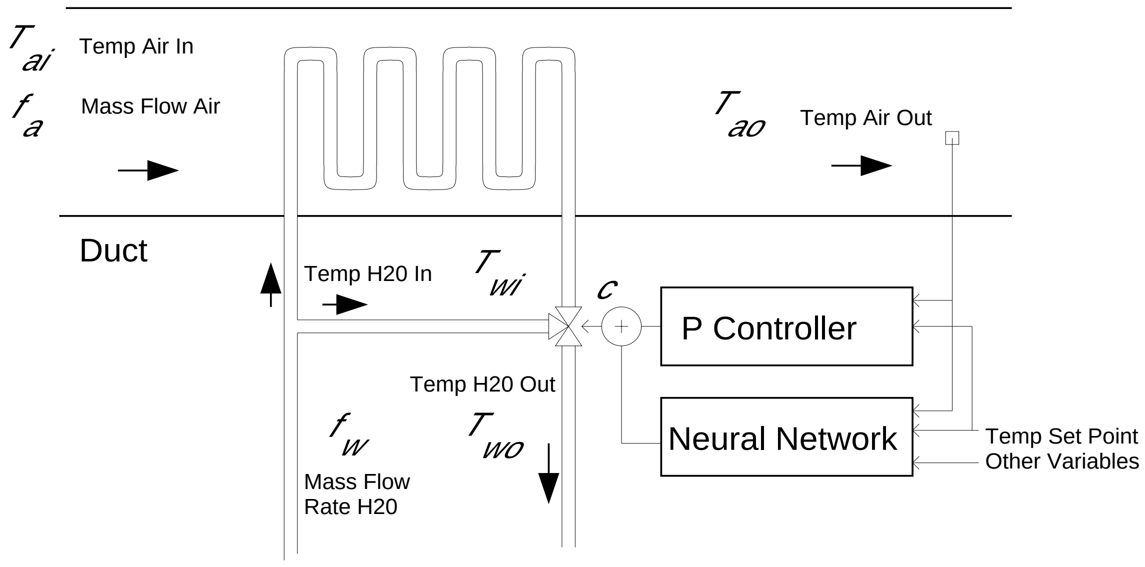


Figure 5: The heating coil controlled by the sum of a proportional controller and a neural network trained to predict the steady-state output of a PI controller.

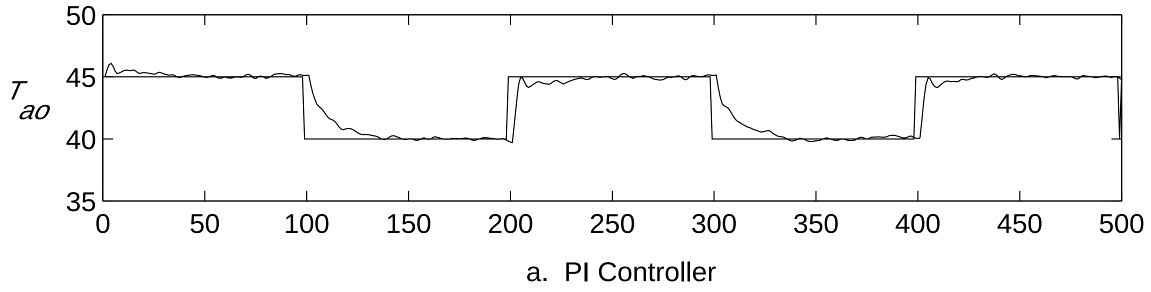
5 Optimal Control with Q-Learning

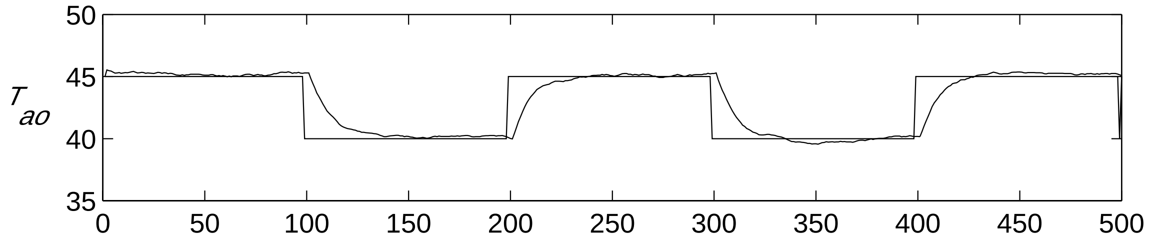
Reinforcement learning methods embody a general Monte Carlo approach to dynamic programming for solving optimal control problems.[3] *Q-learning* procedures converge on value functions for state-action pairs that estimate the expected sum of future reinforcements, which reflect behavior goals that might involve costs, errors, or profits.[11]

To define the Q-learning algorithm, we start by representing a system to be controlled as consisting of a discrete state space, S, and a finite set of actions, A, that can be taken in all states. A *policy* is defined by the probability, $\pi(s_t, a)$, that action a will be taken in state s_t. Let the reinforcement resulting from applying action a_t while the system is in state s_t be $R(s_t, a_t)$. $Q_\pi(s_t, a_t)$ is the value function given state s_t and action a_t, assuming policy π governs action selection from then on. Thus, the desired value of $Q_\pi(s_t, a_t)$ is

$$Q_\pi(s_t, a_t) = \mathrm{E}_\pi \left\{ \sum_{k=0}^{T} \gamma^k R(s_{t+k}, a_{t+k}) \right\},$$

where γ is a discount factor between 0 and 1 that weights reinforcement received sooner more heavily than reinforcement received later. This expression can be rewritten as an immediate reinforcement plus a sum of


a. PI Controller


b. Neural Network Controller

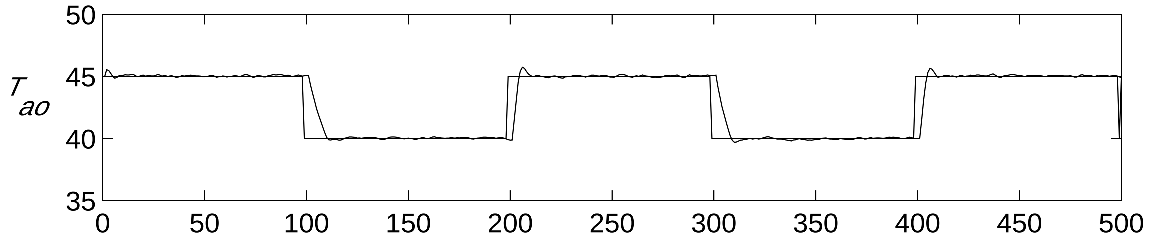

c. Combined Neural Network and P Controller

Figure 6: Set point and actual temperature versus time as the system is controlled with a) the PI controller, b) the neural network, steady-state predictor, and c) the combined neural network predictor and proportional controller.

future reinforcement:

$$\begin{aligned} Q_\pi(s_t,a_t) &= \mathrm{E}_\pi\left\{R(s_t,a_t) + \sum_{k=1}^{T}\gamma^k R(s_{t+k},a_{t+k})\right\}, \\ &= \mathrm{E}_\pi\left\{R(s_t,a_t) + \gamma\sum_{k=0}^{T-1}\gamma^k R(s_{t+k+1},a_{t+k+1})\right\}. \end{aligned}$$

In dynamic programming, policy evaluation is conducted by iteratively updating the value function until it converges on the desired sum. By substituting the estimated value function for the sum in the above equation, the iterative policy evaluation method from dynamic programming results in the following update to the current estimate of the value function:

$$\Delta Q_\pi(s_t,a_t) = \mathrm{E}_\pi\left\{R(s_t,a_t) + \gamma Q_\pi(s_{t+1},a_{t+1})\right\} - Q_\pi(s_t,a_t),$$

where the expectation is taken over possible next states, s_{t+1}, given that the current state is s_t and action a_t was taken. This expectation requires a model of state transition probabilities. If such a model does not exist, a Monte Carlo approach can be used in which the expectation is replaced by a single sample and the value function is updated by a fraction of the difference.

$$\Delta Q_\pi(s_t,a_t) = \alpha_t\left[R(s_t,a_t) + \gamma Q_\pi(s_{t+1},a_{t+1}) - Q_\pi(s_t,a_t)\right],$$

where $0 \le \alpha_t \le 1$. The term within brackets is often referred to as a *temporal-difference* error, defined by Sutton.[9]

To improve the action-selection policy and achieve optimal control, the dynamic programming method called value iteration can be applied. This method combines steps of policy evaluation with policy improvement. Assuming we want to maximize total reinforcement, as would be the case if reinforcements are profits or proximity to a destination, the Monte Carlo version of value iteration for the Q function is

$$\Delta Q_\pi(s_t,a_t) = \alpha_t\left[R(s_t,a_t) + \gamma\max_{a'\in A} Q_\pi(s_{t+1},a') - Q_\pi(s_t,a_t)\right].$$

This is what has become known as the *Q-learning* algorithm. Watkins[11] proves that it does converge to the optimal value function, meaning that selecting the action, a, that maximizes $Q(s_t,a)$ for any state s_t will result in the optimal sum of reinforcement over time. The proof of convergence assumes that the sequence of step sizes α_t satisfies the stochastic approximation conditions $\sum\alpha_t = \infty$ and $\sum\alpha_t^2 < \infty$. It also assumes that every state and action are visited infinitely often.

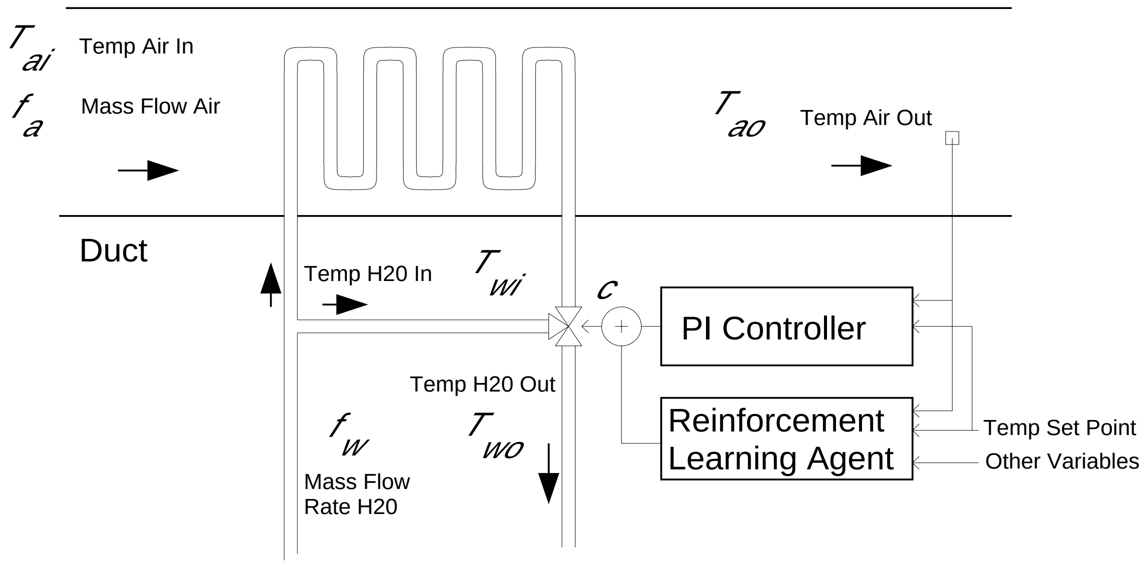


Figure 7: The heating coil controlled by the sum of a PI controller and the action taken by the reinforcement learning agent.

The Q function implicitly defines the policy, π, defined as

$$\pi(s_t) = \underset{a \in A}{\operatorname{argmax}} \, Q(s_t, a).$$

However, as Q is being learned, π will certainly not be an optimal policy. We must introduce a way of forcing a variety of actions from every state. In the following experiment, a random action was taken with probability p_t for step t, where $p_{t+1} = \lambda p_t$, $p_0 = 1$, and $0 < \lambda < 1$. Thus, the value of p_t approaches 0 with time and the policy slowly shifts from a random policy to one determined by the learned Q function.

5.1 Experiment

The reinforcement learning agent was combined with a PI controller as shown in Figure 7. Inputs to the reinforcement-learning agent were T_{ai}, T_{ao}, T_{wi}, T_{wo}, f_a, f_w, and the set point, all at time t. The output of the reinforcement learning agent is directly added to the output of the PI controller. The allowed output actions of the reinforcement learning agent was the set of discrete actions $A = \{-100, -50, -20, -10, 0, 10, 20, 50, 100\}$. The selected action is added to the PI control signal. The Q function is implemented in a quantized, or table look-up, method. Each of the seven input variables are divided into six intervals, which quantize the 7-dimensional input space into 6^7 hypercubes. In each hypercube is stored the Q values for the nine possible actions.

Reinforcement at time step t is determined by the squared error between the controlled variable and its

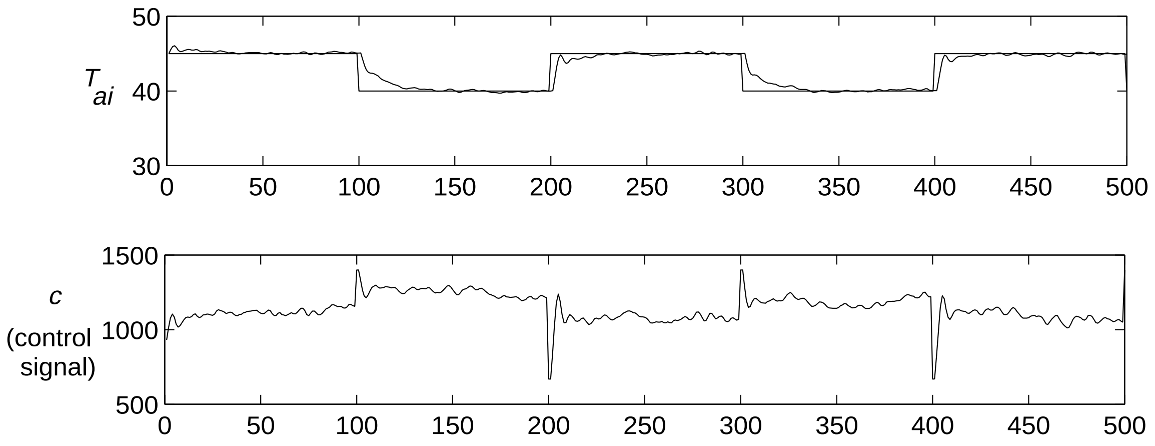


Figure 8: Performance of PI controller. The top graph shows the set point and the actual output air temperature over time. The bottom graph shows the PI output control signal.

set point plus a term proportional to the square of the action change from one time step to the next. Formally, this reinforcement is calculated by

$$R(t) = (T_{ao}(t)^* - T_{ao}(t))^2 + \beta(a_t - a_{t-1})^2,$$

where T_{ao}^* is the set point. The action change term is introduced to reduce the fluctuation in the control signal to minimize the stress on the control mechanism. The values of a_t in this equation are indices of the set A, rather than actual output values.

The reinforcement-learning agent is trained for 1,000 repetitions, called *trials*, of a 500 time-step interaction between the simulated heating coil and the combination of the reinforcement learning agent and the PI controller, hereafter referred to as the RL/PI controller. The performance of the RL/PI controller was compared with the performance of just the PI controller: Figure 8 shows the set point and actual temperatures and the PI control signal over these 500 steps. The RMS error between the set point and actual output air temperature over these 500 steps for just the PI controller is 0.93.

The training algorithm for the reinforcement learning agent depends on the parameters α_t, λ, γ, and β. After trying a small number of values, we fixed γ and β to $\gamma = 0.95$ and $\beta = 0.1$. We tested a number of combinations of values for α and λ, where α is held constant during a training run rather than decreasing it as required for the convergence proofs. Figure 9 shows how performance depends on these parameter values. Performance was measured by the average RMS error over the final 30 trials and also by the average RMS error over all 1,000 trials. The first measure is of final performance and the second is an indication of the speed with which the error was reduced. The best parameter values were $\alpha = 0.1$ and $\lambda = 0.995$. These

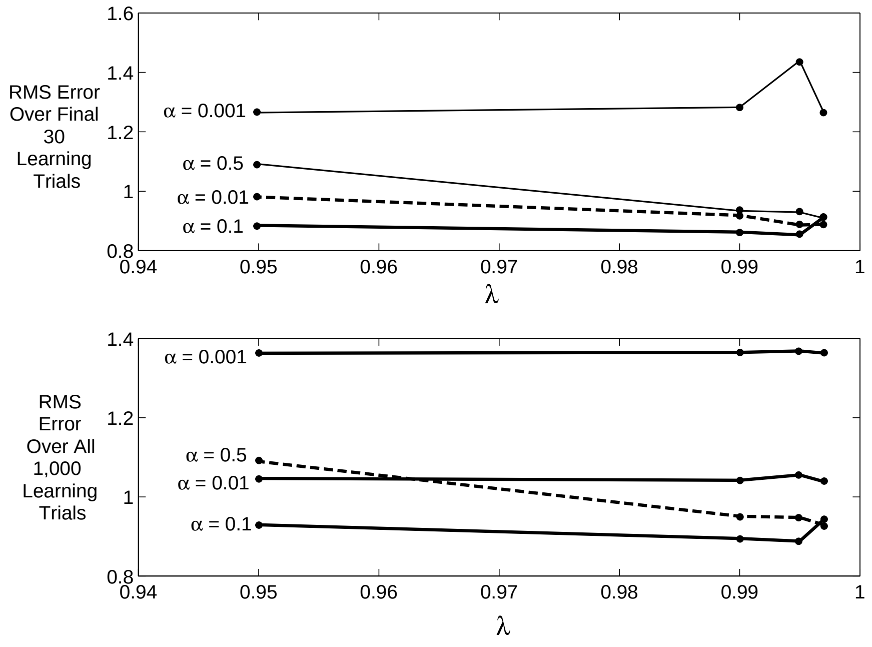


Figure 9: Performance of combined reinforcement-learning agent and PI controller for different parameter values. Top graph shows performance averaged over final 30 learning trials. Bottom graph shows performance averaged over all 1,000 learning trials. Best parameter values are $\alpha = 0.1$ and $\lambda = 0.995$.

values were used to obtain the following results.

5.2 Results

The reduction in RMS error while training the reinforcement-learning agent is shown in Figure 10. After approximately 80 training trials, the average RMS error between the set point and actual temperature was reduced below the level achieved by the PI controller alone. With further training the reinforcement learning agent converges on a deterministic policy that consistently achieves an RMS error of 0.85, about an 8.5% reduction in error from that achieved by the PI controller alone.

The resulting behavior of the controlled air temperature is shown in Figure 11. The middle graph shows the output of the reinforcement-learning agent. It has learned to be silent (an output of 0) for most time steps. It produces large outputs at set point changes, and at several other time steps. The combined reinforcement-learning agent output and PI output is shown in the bottom graph. The reinforcement-learning agent learns to augment the PI controller's output during the initial steps and at most of the set point changes. From time step 340 through about 480 the reinforcement-learning agent injects negative and positive values into the control signal. This pattern of outputs is converged on in most of the successful training runs of the reinforcement-learning agent.

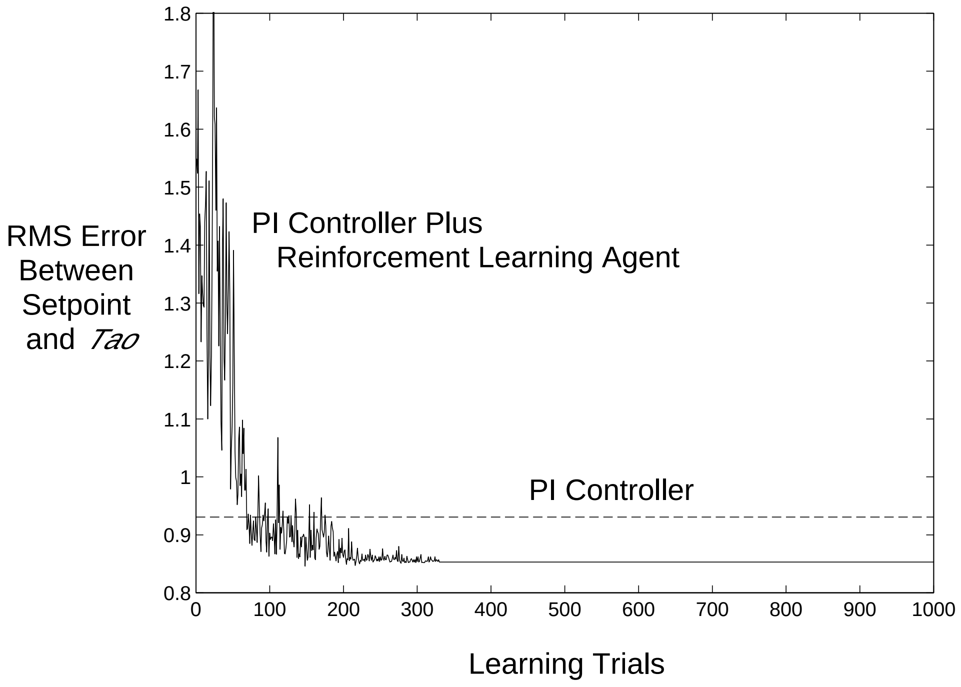


Figure 10: Reduction in error with multiple training epochs. The amount of exploration is reduced during training.

6 Conclusions

Neural networks and reinforcement learning were shown to result in more accurate tracking of the temperature set point of a simulated heating coil. Training was based on a PI controller for the coil. In one experiment, a neural network replaced the PI controller. The other two experiments involved a synthesis of a PI controller and learning agents. The reinforcement learning agent was able to reduce the RMS error of the controlled temperature by about 8.5%. A greater reduction is possible for wider disturbance changes.

Variations of each strategy are being studied further and ways of combining them are being developed. For example, the reinforcement learning agent can be added to any existing control system, including the combined steady-state predictor and P controller. Additional investigations continue on the state representation,[1,2] the set of possible reinforcement-learning agent actions, different neural network architectures, and on additional ways for altering the amount of exploration performed by the reinforcement-learning agent during learning. Plans include the evaluation of this technique on a physical heating coil being controlled with a PC and control hardware.

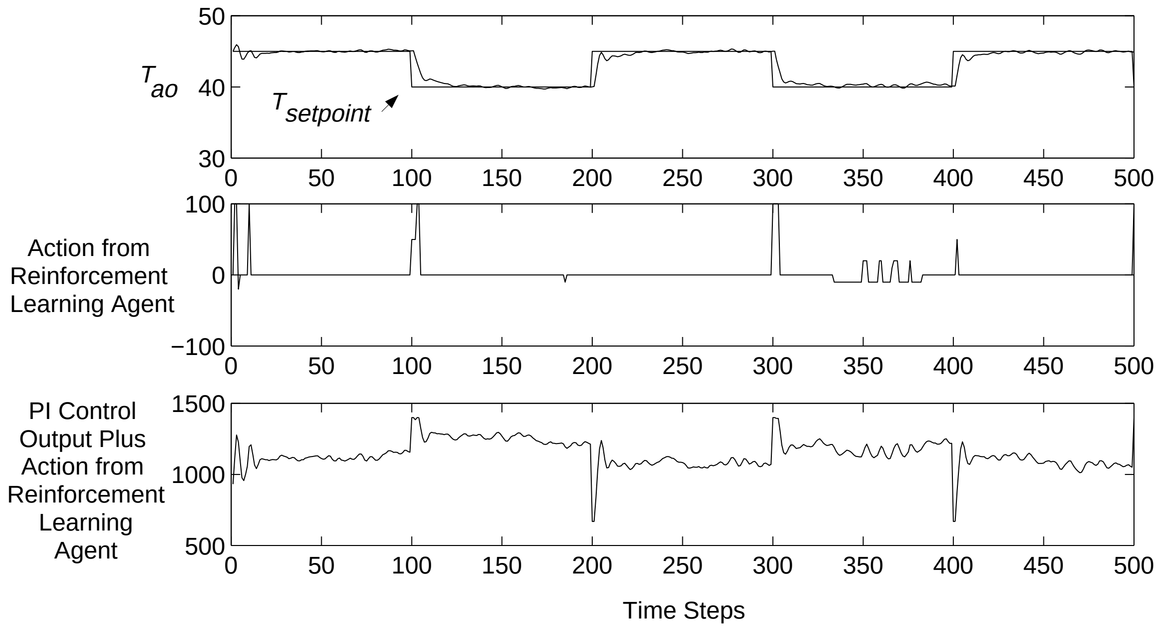


Figure 11: Performance of combined reinforcement-learning agent and PI controller. Top graph shows the set point and the actual output air temperature over time. The second graph shows the output of the reinforcement-learning agent. The third graph shows the combined PI output and reinforcement-learning agent output.

Acknowledgments

This work was supported by the National Science Foundation through grant CMS-9401249.

References

1. Charles W. Anderson. Q-learning with hidden-unit restarting. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 81–88. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

2. Charles W. Anderson and S. G. Crawford-Hines. Multigrid Q-learning. Technical Report CS-94-121, Colorado State University, Fort Collins, CO, 1994.

3. A. G. Barto, R. S. Sutton, and C. Watkins. Learning and sequential decision making. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 539–602. MIT Press, Cambridge, MA, 1990.

4. Peter S. Curtiss. Experimental results from a network-assisted PID controller. *ASHRAE Transactions*, 102(1), 1996.

5. Peter S. Curtiss, Gideon Shavit, and Jan F. Krieder. Neural networks applied to buildings – a tutorial and case studies in prediction and adaptive control. *ASHRAE Transactions*, 102(1), 1996.

6. S. J. Hepworth and A. L. Dexter. Neural control of a non-linear HVAC plant. *Proceedings 3rd IEEE Conference on Control Applications*, 1994.

7. S. J. Hepworth, A. L. Dexter, and S. T. P. Willis. Control of a non-linear heater battery using a neural network. Technical report, University of Oxford, 1993.

8. Minoru Kawashima, Charles E. Dorgan, and John W. Mitchell. Optimizing system control with load prediction by neural networks for an ice-storage system. *ASHRAE Transactions*, 102(1), 1996.

9. R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

10. David M. Underwood and Roy R. Crawford. Dynamic nonlinear modeling of a hot-water-to-air heat exchanger for control applications. *ASHRAE Transactions*, 97(1):149–155, 1991.

11. C. Watkins. *Learning with Delayed Rewards*. PhD thesis, Cambridge University Psychology Department, 1989.