ACCELERATOR AND FEEDBACK CONTROL SIMULATION USING NEURAL NETWORKS*

D. NGUYEN,† M. LEE, R. SASS, H. SHOAEE
Stanford Linear Accelerator Center Stanford University, Stanford CA 94305

Abstract

Unlike present constant model feedback systems, neural networks can adapt as the dynamics of the process changes with time.

Using a process model, the "Accelerator" network is first trained to simulate the dynamics of the beam for a given beam line. This "Accelerator" network is then used to train a second "Controller" network which performs the control function. In simulation, the networks are used to adjust corrector magnets to control the launch angle and position of the beam to keep it on the desired trajectory when the incoming beam is perturbed.

Introduction

Both fast and slow feedback control are used in the SLC control system to stabilize the beam and optimize collisions. Fast feedback control executes at or near the beam rate in the 80386 micros, each of which controls all devices in one geographical region[1], while slow feedback control operates in the VAX over a period of many seconds to a few minutes.

All of these systems use a constant model of the accelerator in the region to be controlled. While known noise characteristics can be incorporated into the model, transient conditions or unknown systematic changes over time can reduce the effectiveness of the control.

Adaptive control systems however, can change their behavior in response to changes in the dynamics of the process they control. Like constant model systems, adaptive systems contain an embedded model of the process they control but they also have the ability to modify this embedded model and the control algorithm with time as the process dynamics change. One way to do adaptive control is by the use of multiple neural networks.

Neural Network Architecture

For this simulation, two neural networks were trained with the back propagation algorithm[2]. Even though a network with only a single hidden layer can implement any nonlinear function[3], multiple networks with more layers often provide a better way to decompose a problem.

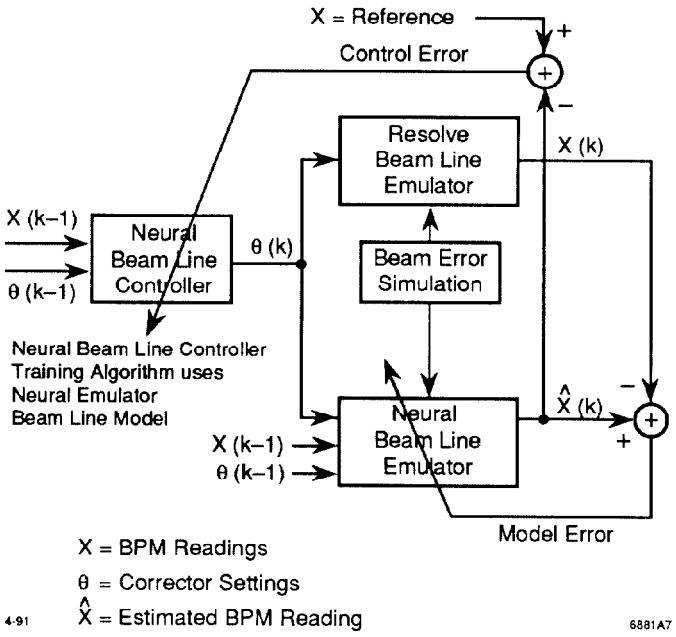

Figure 1. Block Diagram of Neural Network Architecture

Figure 1 shows the neural network architecture used for this simulation which consists of two feed forward networks. The Neural Beam Line Emulator is the machine model, the equivalent of the plant model in control theory, and the Neural Beam Line Controller is the equivalent of the control algorithms used to keep the beam at the desired setpoint.

Because the plant to be controlled is linear, that is, the Beam Position Monitor (BPM) readings for a pulse is a linear function of the corrector settings, both the Neural Emulator and the Controller need only be one layer networks i.e., no hidden layer is required. In a beam line containing 16 BPMs, the Neural Controller is trained to adjust two correctors in each plane near the beginning of the beam line to center the beam positions at two consecutive BPMs near the end of the line.

Training the Networks

The learning process involves two stages. The first stage trains the Neural Emulator to act like the beam line model

*Work supported by the Department of Energy, contract DE-AC03-76SF00515

†Present address: ARGO Systems.

0-7803-0135-8/91$03.00 ©IEEE

and "understand" the beam dynamics in this section of the accelerator. It uses the RESOLVE[4] beam line modeling program as its "teacher". The second stage enables the Neural Controller to learn to control the beam by using the Neural Emulator as a model. It is assumed that the factors which affect the trajectories are varying slowly with time and change little from pulse to pulse, enabling the Neural Controller to use BPM and corrector information from the previous pulse to control the next pulse.

Training the Neural Emulator is similar to plant identification (plant modeling) in control theory, except that the plant identification is done automatically by a neural network. The Neural Emulator has 40 inputs and 32 outputs. The inputs are the 32 BPM readings (16 in each plane) and the four corrector settings (two in each plane) of the previous pulse and the four corrector settings of the current pulse. The Neural Emulator then outputs the 32 BPM readings which are a prediction of the trajectory of the next pulse. The current pulse is then launched, and the BPM readings generated by RESOLVE are compared with the Neural Emulator's prediction. The prediction error is used to train the neural network using the back propagation algorithm so that the Neural Emulator will make a better prediction. After training over many pulses with the correctors set to random values for each pulse, it learns to be a good predictor of the next pulse's trajectory. By this process, the Neural Emulator "gets the feel" of how the corrector settings affect the pulse's trajectory. This process is roughly analogous to the steps that would be taken by a human designer to model the beam line. In this case, however, the modelling is done automatically by the network.

The Neural Controller is also a one layer (no hidden units) network with 36 inputs and four outputs. The inputs are the BPM readings and the corrector settings from the previous pulse, and the outputs are the corrector settings to be used in controlling the next pulse. The Neural Controller's outputs are connected to the RESOLVE beam line model and the Neural Emulator.

Before launching each pulse, the BPM readings and the corrector settings from the previous pulse are obtained and fed to the Neural Controller. Because the adaptive weights of the controller are initially at random, it outputs an erroneous set of corrector settings for the next pulse. The pulse is then launched, and the BPM readings at the two BPMs of interest are obtained. The error, which is the difference between the actual beam positions and the desired beam positions at these two BPMs, is computed, and the controller is trained to minimize the magnitude square of this error. However, to train the controller we need to know the error in the controller's outputs, but we have instead the error in the BPM readings shown in Eq. 1.

$$\epsilon = x_{\text{reference}} - x(k) \tag{1}$$

This is where the Neural Emulator is useful. It essentially acts as an error translator, converting the errors in BPM readings to errors in corrector settings, θ. The error in the BPM readings is back propagated through the Neural Emulator toward the Neural Controller, and then back propagated through the Neural Controller, updating its weights, W, according to the learning rate μ at the same time as shown in Eq. 2.

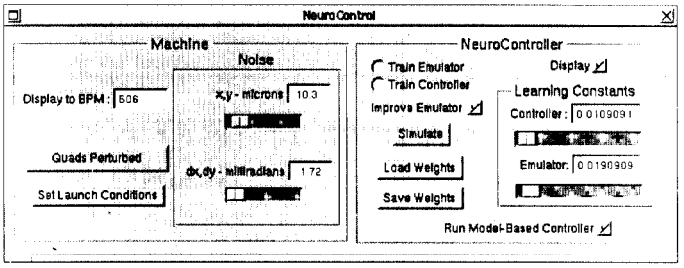

Figure 2. Simulation Control Panel

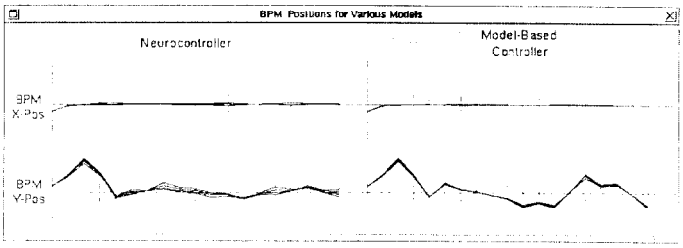

Figure 3. Standard Beamline Environment

$$\Delta W = -\mu \frac{\partial \epsilon^2}{\partial W} = 2\mu\epsilon \frac{\partial x(k)}{\partial \theta(k)} \frac{\partial \theta(k)}{\partial W} \tag{2}$$

The Neural Controller training process is repeated for many pulses, until the mean square error decreases to an acceptable value. The training of the Neural Emulator may continue at the same time as the Neural Controller, thus enabling the whole system to continually adapt. If unexpected changes, such as malfunctions or drifts due to temperature fluctuations, occur in the beam line, the Neural Emulator will adapt to the changes, and then the Neural Controller will adapt to the Neural Emulator, enabling the overall system to continue to function.

Simulation Results

Figure 2 shows the control panel for doing the simulation. The learning constants for both of the networks, noise values of beam position and angle, which of the networks are to be trained and comparison with the model based controller can all be selected from this panel. In addition, the "Quads" button allows us to alter predesignated quadrupole strengths and thus invalidate the machine model.

Figure 3 shows a comparison of the trajectories of a series of consecutive pulses controlled by the neural networks and the model based controller. Note how the pulses come into the beam line section off center and at an angle, and are centered by the end of the beam line section. With all devices matching the model, both the model based controller and the neural networks work about equally well.

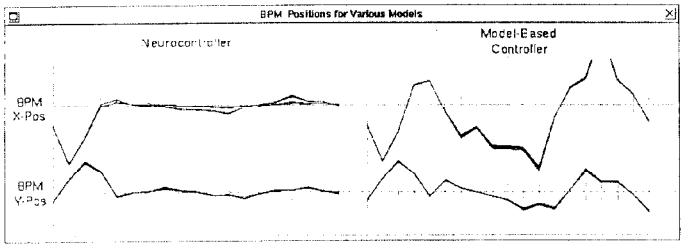

Figure 4. Malfunctioning Beamline Environment

Figure 4 shows the simulation in a beamline environment which contains two quadrupoles whose field strength values have been substantially decreased from their nominal values, simulating a malfunction. Because the model used does not match the actual beam line due to the simulated malfunction, the beam is not well controlled at all by the model based controller. The neural network based controller however has adapted to the change and still controls the beam as well as before, after it has learned how the real machine has changed. When the quadrupoles have been restored to their nominal values, the neural network based controller can again relearn the correct model and control the beam like the model based controller.

Future Plans

Neural network based feedback can be used to compensate for changes in the energy profile caused by Klystron population changes, Klystron phase drift and diurnal effects not compensated for by existing feedback. Less likely though still possible are hardware changes like power supply instabilities and changes in focusing properties of magnets due to hysteresis. All of these dynamic changes to some degree invalidate the static model used by current feedback systems.

To further explore the possibilities for the use of neural networks in the real control environment we anticipate attempting one or more of the following efforts:

1. Train one or more emulators offline using the models generated for the fast feedback system presently running at the SLC[1]. Then run these neural network models in the 80386 micros that presently run the fast feedback system and compare the results with the existing feedback.
2. Use a neural network to control the residual errors of the existing feedback systems. In this mode, the network acts to "touch up" i.e. make small, dynamic corrections to the existing feedback system.
3. In parallel with this work, experiment with a variety of extensions and enhancements to the basic back propagation algorithm to improve performance and study network instability.

Conclusions

We have been able to train multiple neural networks to be both an emulator and controller of a beam line section. This indicates that it would be feasible to use multiple neural networks to compensate for transient or slowly varying beam instabilities or correct for uncertainties in our knowledge of the machine model.

Acknowledgements

Special thanks to Professor Bernard Widrow of Stanford University for his early inspiration on this project.

References

[1] F.R. Rouse et al., "General, Database Driven Fast Feedback System for the Stanford Linear Collider," *Proceedings of the 1991 IEEE Particle Accelerator Conference*, San Francisco, CA (May 1991).

[2] G.E. Hinton, D.E. Rumelhart and R.J. Williams. *Learning internal representations by error propagation*, Vol. **1**, Ch. 8. MIT Press, Cambridge, MA, (1986).

[3] B. Irie and S. Miyake. "Capabilities of three-layered perceptrons," *Proceedings of the IEEE International Conference on Neural Networks*, pp. I-641, (1988).

[4] Cf. M. Lee for information on RESOLVE.