System Identification with AR model

Enes Erçin
Faculty of Electrical Engineering
Istanbul Technical University

1. Introduction

This paper aims to show a practical approach to the PID tuning problem. Due to the complications caused by choosing PID coefficients for the different performance parameters various tuning methodologies are discussed. In this paper deep neural network structure is used to tune the PID controller in addition application of recurrent neural network is introduced. Study is inspired by different resarchers mainly S. Akhyar's "Self-Tuning PID Control by Neural Networks"[1] paper.

The initial goal was to exploit already available frameworks such as Tensorflow or Matlab's toolbox, yet it does not seem possible for this unique application. The weight update methodology is unorthodox. Usually, there are inputs and outputs of the neural networks and we would expect desired output to come from an outside system. This application does not include outside reference coefficients for the PID controller. It is not provided and for a good reason. For most cases, the PID controller's optimum coefficients are unknown thus if the desired coefficients already existed there would be no need to use neural networks to identify those coefficients.

Aim is to deals with two kinds of systems linear and non-linear. The purpose of this variety is to evaluate the practicality of the application of recurrent neural networks also deep neural networks for different types of systems. It has been mentioned in Akhyar's paper that deep neural networks can indeed work on such systems for both non-linear and linear [1], this paper focuses on hands-on applications from scratch. Due to the practicality of this study pieces of code will be shared right after the theoretical background is introduced.

2. System Description

In order to describe the weight update methodology to the deep neural network, the first system model should be introduced. As mentioned we are dealing with two different systems one which has a linear plant output and the other is non-linear. The plant's process is described with given formulations 1 and 2.

$$y[n+1] = 0.988y[n] + 0.232u[n] \quad (1)$$

$$y[n+1] = 0.9y[n] - 0.001y[n-1]^2 + u[n] + sin(u[n-1]) \quad (2)$$

Plants input u[n] is generated from the PID controller. The formulation of the PID controller can be described with discrete and continuous values.

2.1. PID Controller

The system we are dealing with includes a PID controller that has the task of adjusting the input signal to the plant using the previous error signal. The PID controller is composed of three sub-modules those are proportional, derivative, and integral. Generally, PID controller sub-modules are connected in parallel then the results of each sub-module are added and transferred to the plant as an input.

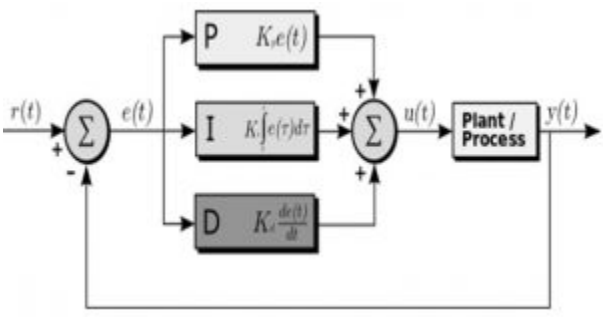

Figure 1. PID controller

PID controllers can work both in discrete and continuous inputs. Although the neural network has to work with discrete sequences it is better to use a discrete PID controller for consistency. Figure [1] shows a continuous application of PID. To make a PID controller work in discrete inputs better approach is to use numeric methods of integration and differentiation.

2.1.1. Numeric Integration.

T. here are multiple different ways to take an integral with discrete inputs most well knowns are Midpoint and Trapezoidal approaches. The reasoning's behind calculations are just as their name suggests. Midpoint assumes the data points are the same as the middle points of two data between sampling periods. The addition of each rectangular is the result of integral by numeric calculation using a midpoint application [2]. However, this paper utilizes a trapezoidal rule, similar to the midpoint this is also done by calculating an area of well-known shape between sampling periods. Two

points are at each end of the trapezoidal and the sampling period is the height of it.

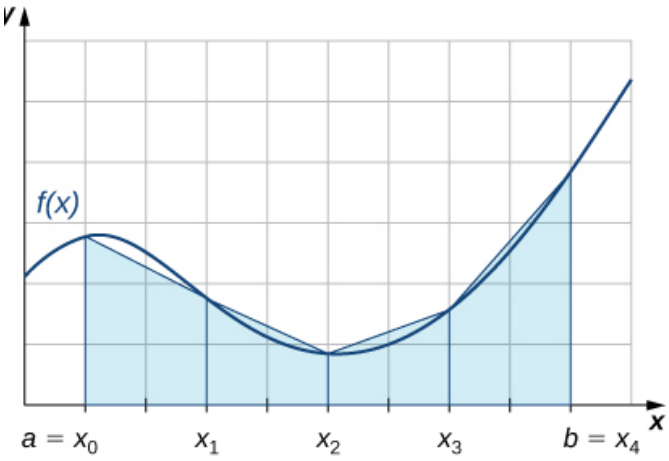

Figure 2. Trapozoid Integration

Numeric integration can be coded as this.

```
class Intg():
    def __init__(self,sr):
        self.prev_val = 0
        self.acc = 0
        self.sr = sr
    def proc_(self,new_val):
        self.acc = self.acc + (self.sr*0.5)*(self.prev_val + new_val)
        prev_val = new_val
```

2.1.2. Numeric Differentaiton.

C. ontinuous differentiation comes from applying a difference of two points while the sampling period goes to infinity. Mathematically differentiation is represented as this formulation 3.

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3}$$

Considering this numeric differentiation for discrete values sampling period is not incrementally small but rather a different value. So numeric differentiation can be coded as this.

```
class Dif_():
    def __init__(self):
        self.prev_val = 0
    def proc_(self,new_val):
        self.acc = (new_val- self.prev_val)/(self.sr)
        prev_val = new_val
```

Finally, when we use integrator, differentiator, and proportional calculations we can establish a PID controller which can function in a discreete time state.

```
class PID_CNTRL():
    def __init__(self,Kp = 0.9,Kd = 0.4 ,Ki = 0.2 ):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd

        self.INTG_BLOCK = Intg_num(1,0)
        self.Diff_num = Diff_num(1,0)

    def _proc(self,new_e):
        intg_res = self.INTG_BLOCK.intg(new_e)
        dif_res  = self.Diff_num.diff(new_e)

        res = self.Kd*dif_res + self.Ki*intg_res + self.Kp*new_e

        return res
```

After defining PID with discreet time now we can formulate the u[n] which is used as input to the plant process.

$$u[n] = Kp*e[n] + Kd*(e[n] - e[n-1]) + Ki*\sum_{i=0}^{n} e(i) \tag{4}$$

$$e[n+1] = u[n] - r[n] \tag{5}$$

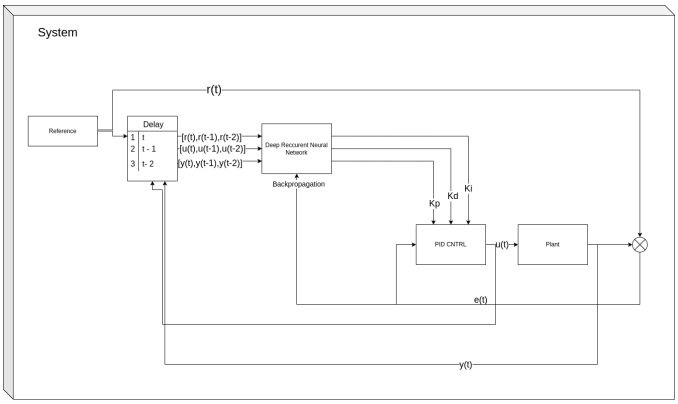

Figure 3. System Block Diagram

3. Backpropagation

Since there is no reference inputs for PID coefficients, there has to be another approach to relate the weigth factors to the error result. Regular neural network would use backpopagation with the error J.

$$J = 0.5(ref - out)^2 \tag{6}$$

$$\delta^{(L)} = \frac{\partial J}{\partial a^{(L)}} \odot g'(z^{(L)})$$

$$\frac{\partial J}{\partial W^{(L)}} = \delta^{(L)} \cdot a^{(L-1)T}$$

$$\frac{\partial J}{\partial b^{(L)}} = \delta^{(L)}$$

$$\delta^{(l-1)} = (W^{(l)T} \cdot \delta^{(l)}) \odot g'(z^{(l-1)})$$

One particular problem in this experiment is that the back-propagation has to be done with the errors between reference point and output of the plant. Same signal which PID controller is provided with. To solve this problem we extend the gradient decent calculations. Partial derivative of the system output should be linked with the output of PID contoller.

$$\delta^{(L)} = \frac{\partial J}{\partial y_{n+1}} \cdot \frac{\partial y_{n+1}}{\partial y_n} \cdot \begin{bmatrix} \frac{\partial u_n}{\partial K_p} & \frac{\partial u_n}{\partial K_d} & \frac{\partial u_n}{\partial K_i} \end{bmatrix}^T \quad (7)$$

After derivation of each element of the chain rule we get the following for linear system.

$$\frac{\partial J}{\partial u_n} = -(r[t] - u[t]) = -e[t+1] \quad (8)$$

$$\frac{\partial u_n}{\partial y_{n+1}} = 0.988 \quad (9)$$

$$\begin{bmatrix} \frac{\partial u_n}{\partial K_p} & \frac{\partial u_n}{\partial K_d} & \frac{\partial u_n}{\partial K_i} \end{bmatrix}^T = [e[t]\, e[t]\, e[t]]^T \quad (10)$$

In conclusion of partial derivates the we derived direct relation of neural network layers to the system error. Insted of using the equation 6 we can use $-e[t+1] * 0.988 * [e[t]\, e[t]\, e[t]]^T$.

$$-e[t+1] * 0.988 * [e[t]\, e[t]\, e[t]]^T = \frac{\partial J}{\partial K[n]} \quad (11)$$

$$\frac{\partial J}{\partial K[t]} * \frac{\partial K[n]}{\partial W[n]} = \frac{\partial J}{\partial W[n]} \quad (12)$$

4. Choice of parameter

Since there are a lot of different parameters to get a certain performance the best approach is to test multiple different factors. One such factor is the activation layer. In Akhyar's research sigmoid is the chosen activation function. Due to the poor performance which I have recognized in my application the non-linear activation layer chosen in this application is rectified linear unit function. The dept of the neural network did not have much effect on it is chosen arbitrarily and can be modified if needed. The only restriction is that the output layer must have three nodes as there are three outputs and there should be 12 number nodes in the input layer as the figure 3 suggests. The reason for this is to recognize a possible hidden state existing in time. From each signal stream, there is a memory of the interface which stores three past values and sends the values to the neural network. The streams of signals are reference signals, error signals, plant output, and PID output signals. The learning rate of the neural network has a considerable impact on the system. High learning rates may cause the system to be unstable while low learning rates do cause a decrease in improvements.

5. RNN VS Deep neural network

The difference between regular deep neural networks and recurrent neural networks is simple. On the algorithmic side same weights and biases are used over again. The iteration count is the same as the number of input layers chosen. For example $\begin{bmatrix} Xt & Xt-1 & Xt-2 \end{bmatrix}$ takes three iterations. Each hidden layer takes two inputs, one coming from the stored previous value and the other from the output of the previous layer $\begin{bmatrix} Y1 & Y2 \end{bmatrix}$. It is possible to give an array of input to the node and likewise to output an array of signals. This means that Xt and Y1 can be also an array $\begin{bmatrix} Y11 & Y12 & Y13 \end{bmatrix}$, the only limitation is that input arrays and output arrays may have the same number of elements in them. It has been mentioned that neural network structures have a variety of different parameters. One approach is to add a regular neural network layer to the recurrent neural network layer but this would greatly complicate backpropagation calculations thus it is discarded.

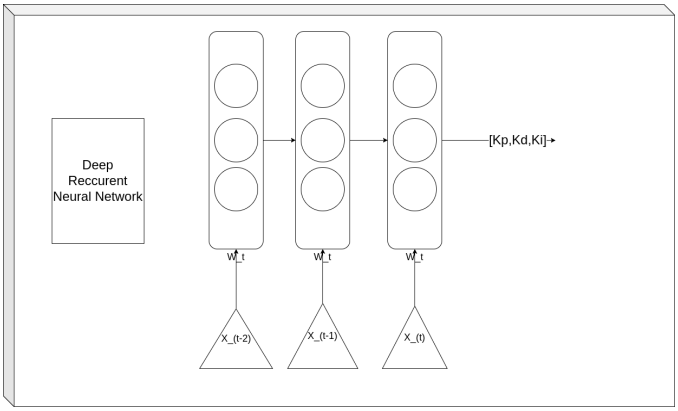

Figure 4. System Block Diagram

6. Implementation

Due to the fact coefficients of the PIDs are not provided externally we had to design the neural networks from scratch. Since frameworks such as Tensorflow or Matlab's tool box do not provide a built-in function to update the parameters in extended backpropagation methods with partial differentials. The choice of codding language is python due to easy matrix calculations and graphing tools such as numpy and matplotlib's which are free to use and open source code libaries. The code of PID is alread given as an example. Like pid deep neural networks and plants will also be in the form of classes. Classes are objects in Python, which carries their own methods and also stores them on specific type of variables.

A deep neuron network object creates neuron network structures, by given parameters. The parameter layer width is an array of integers that indicates the number of neurons at each layer. For our application, the last element of the layer width must be 3 and the first input of the layer width

must be 12. Other layers can be chosen arbitrarily. The learning rate should be between 1-0 but choosing a smaller value is generally a good practice for this application also. Just like pid coefficients neuron network layers' learning rate is hard to tune perfectly.

```
class DeepNN():
    def __init__(self,layer_width,learning_rate,update_method = "err"):
```

7. Conclusion

The outcome of the experiment shows that it is possible to improve the system by improving the accuracy up to 10% on average via adjusting the PID coefficients by the neural network on a linear system however this is not the case for nonlinear systems. It is possible to test the simulation by adjusting different parameters like layer width, neuron count, learning rate, etc. The most crucial elements of implementation is that the backpropagation derived in this experiment is not entirely correct. In the equation 10 we assumed pid $\frac{\partial u_n}{\partial Kd}$ is only related with e[n] and e[n-1] is dismissed in calculations. Which is not the case. As the e[n-1] has some contribution to the partial derivative but it is not easy to calculate and has less importance on the result. In the future, the correct derivation of backpropagation will be investigated. To cover up this issue neural network outputs of coefficients are limited to certain restrictions. To keep away from unstable conditions. It is important to mention that this is a workaround problem and not a permanent solution.

List of Resources

1) Saiful Akhyar, Sigeru Omatu (1993). Self-Tuning PID Control by Neural Networks *International Joint Conference on Neural Networks*
2) MathLibretexts: "Numerical Integration - Midpoint, Trapezoid, Simpson's Rule"
3) Check on: https://github.com/EnesErcin/NeuralNetwork

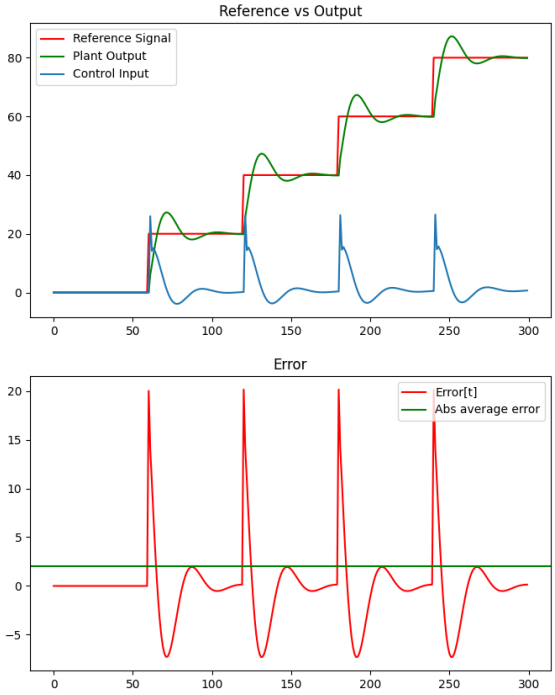

Figure 5. Linear system without neural network

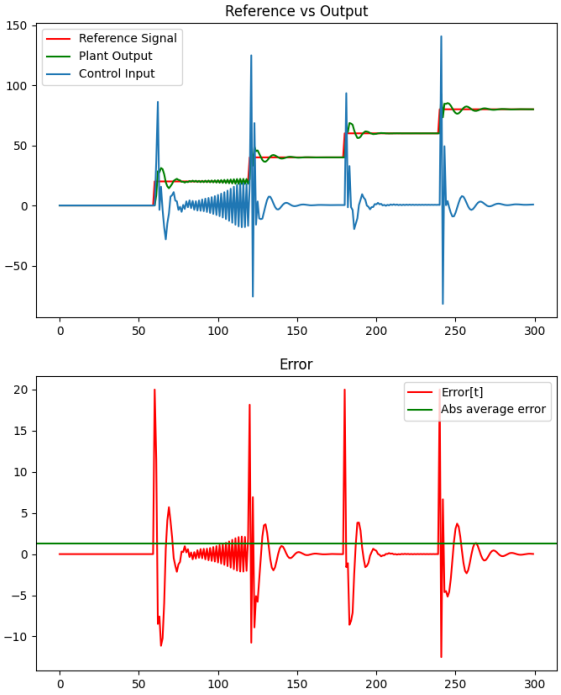

Figure 6. Linear system with neural network

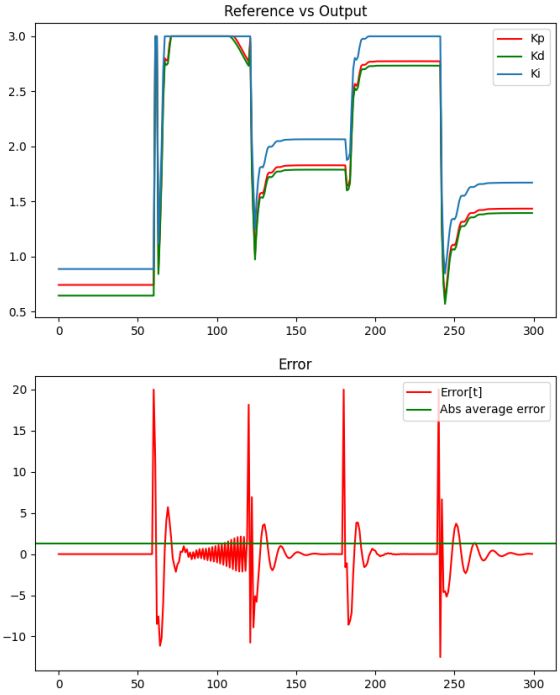

Figure 7. Coefficent update