# 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.

$$y[n+1] = 0.988y[n] + 0.232u[n] \tag{1}$$

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

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.

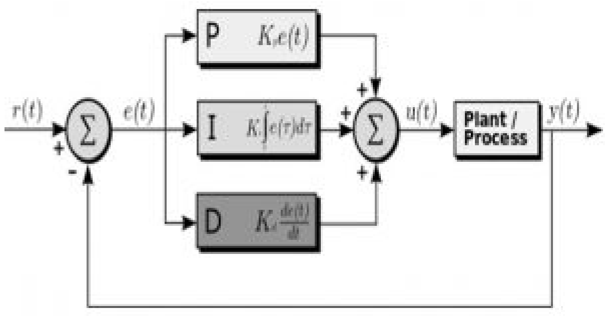

Figure 1. PID controller

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.

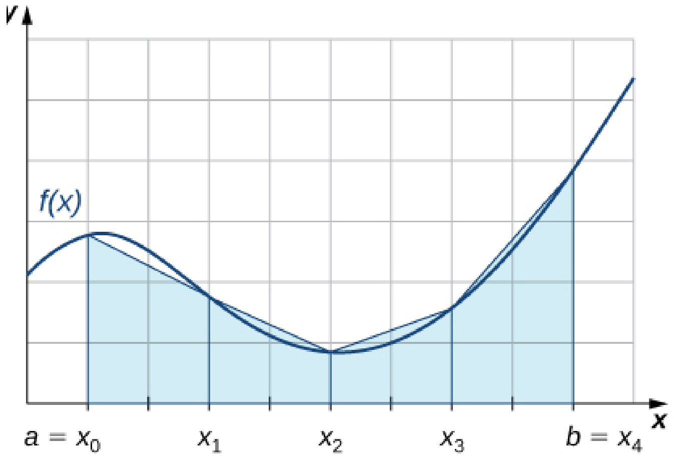

Figure 2. Trapozoid Integration

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.

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

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

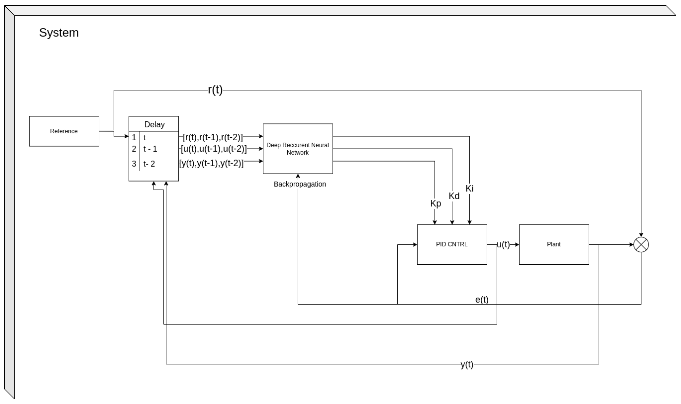

Figure 3. System Block Diagram

## 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.

$$J = 0.5(ref - out)^2$$

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

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

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

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

Since we do not have J directly corresponding to the weigths we can use same approach of gradient decent method.

$$\delta^{(L)} = \frac{\partial J}{\partial y_n + 1} * \frac{\partial y_{n+1}}{\partial y_n} * \frac{\partial y_n}{\partial K_p, \frac{\partial y_n}{\partial K_d}, \frac{\partial y_n}{\partial K_i}]T}$$

## 4. Evaluation

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.

## 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 [Xt, Xt-1, Xt-2] 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 [Y1, Y2]. 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 [Y11, Y12, Y13], 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.

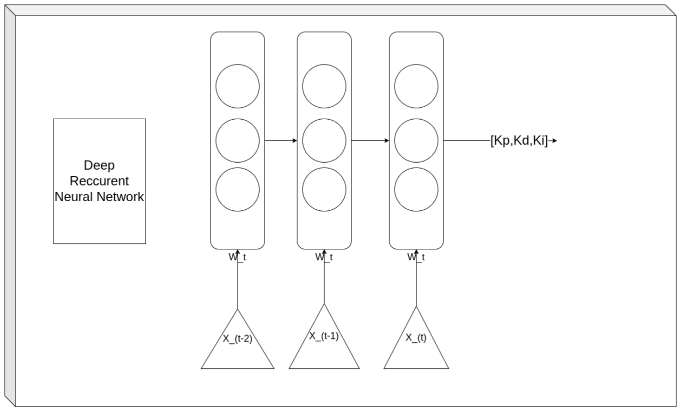

Figure 4. System Block Diagram

## 6. Implementation

The comparison of the identification models and algorithms of recurrent neural network and auto regressive model with the given parameters and described data shows that recurrent neural network has a better performance than the AR model. It is important to mentioned that performance of this study only focuses on these parameters each change can impact the result substantially. Although the outcome can also depends on the iterations as not all iterations converge to the same weights or coefficients it is important to give one trail results to consider the differences of the performances numerically. At the end of the training mean of absolute sum of the adapted system errors of AR model $5.3 \cdot 10^-2$ while the mean of sum of adapted system errors of RNN model came out to be $3.4 \cdot 10^-2$. The actual error of the system without any identification algorithm is $8 \cdot 10^-2$. Keep in mind that error is generated by using random function thus there can be slight differences between each iteration as well as the weights of the neural network. These small differences can significantly effect the converged values of the weights.

## 7. Conclusion

The outcome of the experiment shows that it is possible to improve the system by improving the accuracy up to 30% on average via adjusting the pid coeeficents by neural network ON LİNEAR SYSTEM HOWEVER THİS İS NOT the case for nonlinear system. It is possible to test the simulation by adjusting different parameterss like layer width, nueron count, learning rate etc.

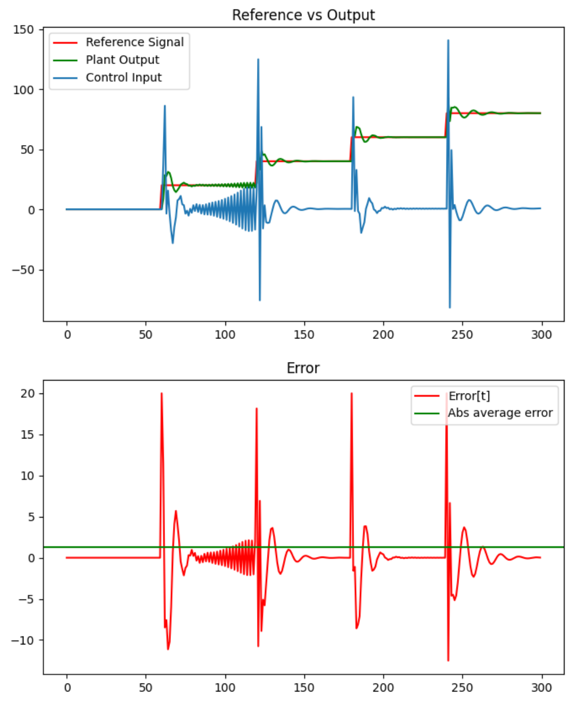

Figure 5. System Block Diagram

Check on: https://github.com/EnesErcin/NeuralNetworkSystemID.git

## List of Resources

1) Saiful Akhyar, Sigeru Omatu (1993). Self-Tuning PID Control by Neural Networks *International*

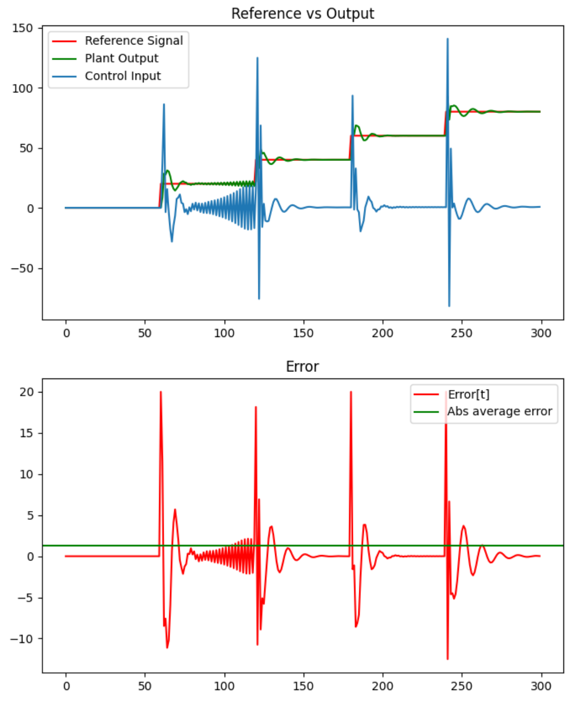

Figure 6. System Block Diagram

*Joint Conference on Neural Networks*

2) MathLibretexts: ”Numerical Integration - Midpoint, Trapezoid, Simpson’s Rule”