11 Linear Sequence Processing

Objective

Neural network inputs can take several different forms. In previous chapters we considered networks with vector inputs (tabular form) and image inputs. In this chapter we consider networks with sequential inputs. In particular, we will focus on networks that perform linear operations on sequences. By beginning with linear operations, we can focus on some basic concepts that will be important when progressing to more complex networks, which we cover in later chapters.

Theory and Examples

What are sequences?

A *sequence* is an ordered collection of elements, where both their values and their order matter. Sequences can appear in a variety of application areas, as you can see in the following table.

Application	**Example Sequence**
DNA analysis (nucleotides)	{A, T, C, G, G, T, A, C, C, C, T, T}
Protein folding (amino acids)	{Met, Gly, Trp, Ser, Cys, Ile, Ala, Leu, Phe, Leu}
E-commerce (actions)	{homepage, product search, category browse, product view, add to cart, view cart, start checkout, payment, confirmation}
Stock predition (prices)	{150.25, 151.50, 153.15, 155,75, 154.30}
Health monitoring (heart rate)	{72, 73, 75, 74, 75, 73, 73, 76, 74}
Translation (words)	{'A', 'sequence', 'is', 'an', 'ordered', 'set', 'of', 'elements'}

The elements in the sequence do not have to be numerical, but before the sequence can be input to a neural network it must be converted to a real number, or a vector of real numbers, in an embedding process. (We will discuss this in later chapters.) For now, we will assume that all network inputs are numerical. We will refer to individual elements of a sequence as either *time points*, or *positions*.

Processing sequences is fundamentally different than processing static examples, in which each example input has no relationship to other examples and can be processed alone without losing any important information. Also, adding temporal (or positional) dimensions creates several cognitive challenges. For example, static problems can be visualized at once, whereas temporal problems require "playing through" sequences, and each time step adds dimensions to the problem space. In addition, sequential processing models can have feedback loops, which require circular (chicken and egg) reasoning.

Dynamic systems

In order to process sequences effectively, we need systems with memory – *dynamic systems*. If a system had no memory it would be a static function mapping inputs to outputs, and the output would be determined solely by the inputs at the the same time point. Dynamic systems are characterized by a state that evolves over time, and the system output depends on both the current input and the state.

Chapters 10 and 14 of NND2 also describe dynamic neural networks.

Some dynamic systems have finite memory, while others have infinite memory. Infinite memory requires that the system have feedback (recurrent connections). In this chapter we consider only *linear*, *discrete-time* systems. We will start with finite memory systems, and then we will extend the conversation to infinite memory systems.

For *linear* systems, if the input is doubled, then the output is doubled. If two inputs are added together, the total output will be the sum of the two individual outputs.

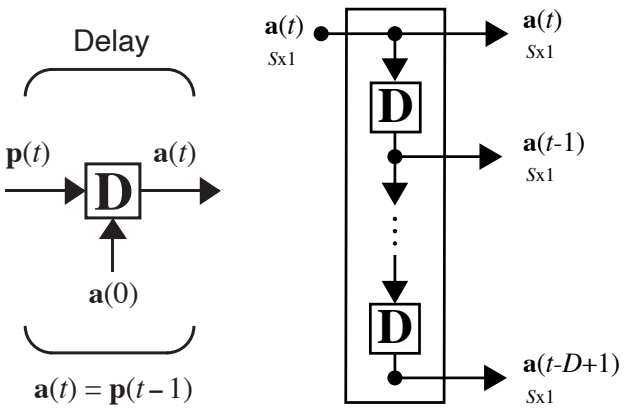
Memory is incorporated into dynamic networks using the *delay*, which we first introduced in Chapter 2 of NND2. The output of the delay is computed from its input according to

$$\mathbf{a}(t) = \mathbf{p}(t-1) \tag{11.1}$$

Discrete-time systems operate on an indexed sequence of numbers, as opposed to continuous-time systems, which operate on continuous input signals. There are continous-time neural networks, but we don't cover those in this text.

Note that in Figure 11.1a the initial condition for the delay, $\mathbf{a}(0)$, is indicated by the vertical arrow going into the delay block.

As described in Chapter 10 of NND2, it is often useful to stack delays together to obtain a *tapped delay line* (TDL). This produces a history of previous values of the input, as shown in Figure 11.1b.



(a) Delay (b) Tapped Delay Line

Figure 11.1: Delay and Tapped Delay Line

Differences between static and dynamic problems

Since all of the previous chapters have focused on static problems, it is worth taking a moment to reset our thinking to focus on dynamic problems. In static problems the neural network is, in essence, approximating a static function. It could be a static function that maps from characteristics of a neighborhood to the price of a home, or it could be a static function that maps from pixel values to the probability that the image contains a cat. In dynamic problems, however, the neural network is approximating a dynamic system, which maps from input sequences to output sequences.

To illustrate this idea, consider the the blue dots in Figure 11.2. They could be targets for a static network whose inputs range from 0 to 10. In that case the network would be approximating the static function $y = f(p) = \sin(2\pi p/10)$, and each blue dot is at a different value of p.

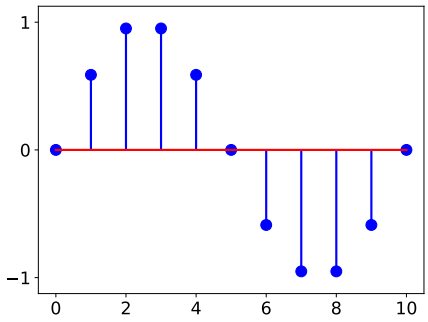


Figure 11.2: Sinusoidal Function

For the static problem, Figure 11.3 would provide a solution, where the weights and biases are set to the values given in Eqs. 11.2 and 11.3.

$$\mathbf{W}^1 = \begin{bmatrix} -0.6246 \\ -0.5410 \\ -0.6246 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} 0.3239 \\ 2.7051 \\ 5.9225 \end{bmatrix} \tag{11.2}$$

$$\mathbf{W}^2 = \begin{bmatrix} -5.6565 & 7.4708 & -5.6565 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} 1.9211 \end{bmatrix} \tag{11.3}$$

On the other hand, the blue dots could represent a time sequence, produced by passing the input sequence in Figure 11.4 through a dynamic system. In that case, our network would be approximating the dynamic system $y(t) = 1.62y(t-1) - y(t-2) + 0.6p(t-1)$, and each blue dot is at a different value of t.

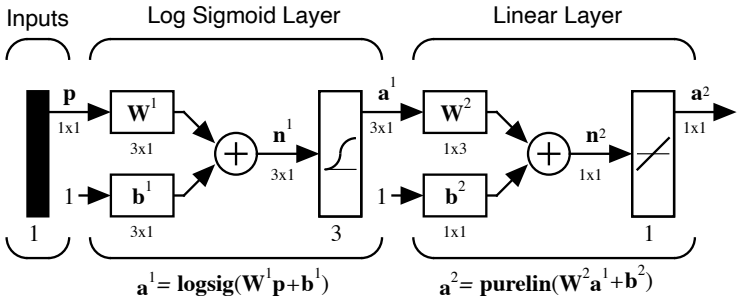


Figure 11.3: Multilayer Network Solution to the Static Problem.

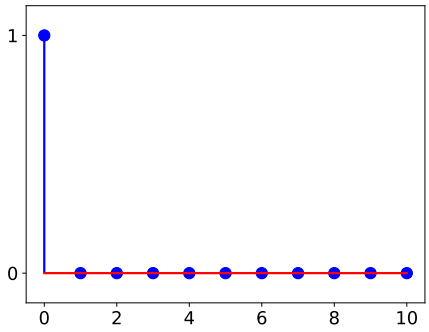


Figure 11.4: Input Sequence

For the dynamic problem, the network in Figure 11.5 would provide a solution, with the weights set to the values given in Eq. 11.4.

$$\mathbf{IW} = \begin{bmatrix} 0.6 \end{bmatrix}, b = \begin{bmatrix} 0 \end{bmatrix}, \mathbf{LW} = \begin{bmatrix} 1.62 & -1 \end{bmatrix} \tag{11.4}$$

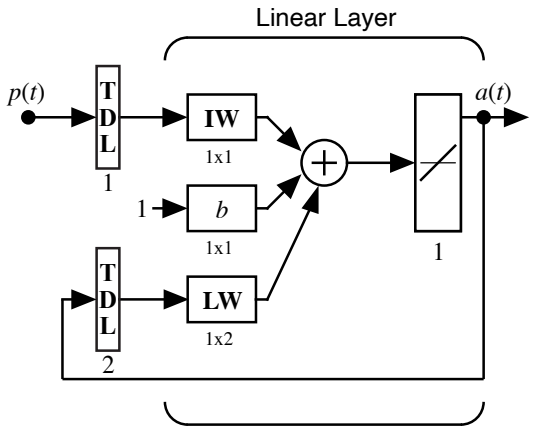


Figure 11.5: Dynamic Network Solution to the Dynamic Problem

To summarize, here are some differences to keep in mind when approaching static and dynamic problems:

- Static problems:

  - Example inputs are single scalars, vectors, arrays or tensors.
  - The network has no memory.
  - Each example input is considered in isolation.
  - Static problems use algebraic equations.
  - Static networks don't have feedback
- Dynamic sequence processing problems:
  - Example inputs are sequences of scalars, vectors, arrays or tensors.
  - The network has memory.
  - Each time step adds dimensions to the problem space.
  - The network must use relationships between the elements of each sequence.
  - Effects can be separated from causes by many time steps.
  - Dynamic problems require difference equations.
  - Dynamic networks may use feedback.
  - Feedback creates stability issues.

Example: Sequence averaging

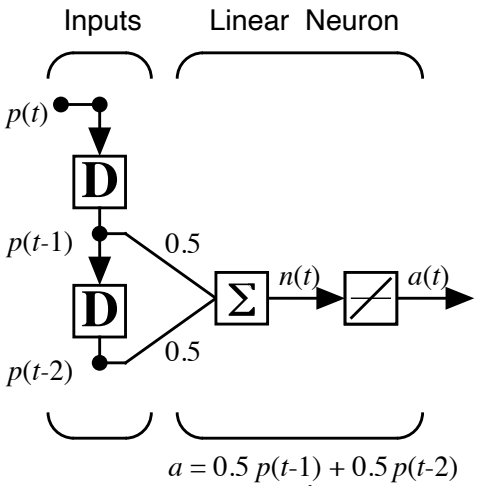


Figure 11.6: Sequence Averaging Network

Figure 11.6 is an example of a dynamic sequence processing network. A TDL is used at the input of the network to store the previous two time points of the input sequence $p(t)$. The two weights in the network are both equal to 0.5, so the network averages the last two values of the sequence.

Table 11.1 illustrates how the network would respond to a particular input sequence, where each column of the table corresponds to a different time point. Figure 11.7 shows plots of the input and output sequence.

Table 11.1: Sequence Averaging Network Operation

t	0	1	2	3	4	5	6	7	8
$p(t)$	0	1	2	3	2	1	0	0	0
$a(t)$	0	0	$\frac{1}{2}$	$\frac{3}{2}$	$\frac{5}{2}$	$\frac{5}{2}$	$\frac{3}{2}$	$\frac{1}{2}$	0
TDL	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}2\\1\end{bmatrix}$	$\begin{bmatrix}3\\2\end{bmatrix}$	$\begin{bmatrix}2\\3\end{bmatrix}$	$\begin{bmatrix}1\\2\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$

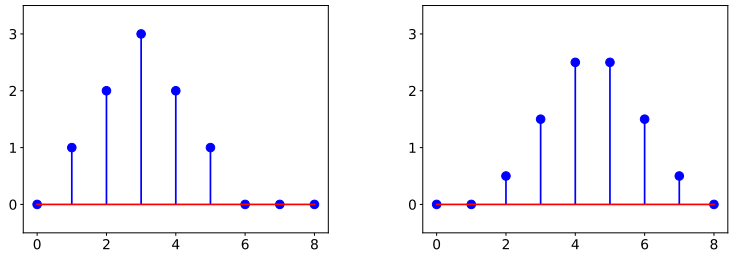


Figure 11.7: Sequence Averaging Network Response

To experiment with this sequence averaging network, use the Deep Learning Demonstration Sequence Averaging Network (**`dl11san`**).

The following code shows how to implement the sequence averaging network in Python.

```
class averaging_network:
   def __init__(self, iw, p_tdl=[]):
       self.iw = np.array(iw)

      if len(p_tdl) > 0:
          self.p_tdl = np.array(p_tdl)
       else:
          if len(self.iw) > 0:
              self.p_tdl = np.zeros(len(self.iw) - 1)
          else:
              self.p_tdl = np.zeros(0)

    def step(self, p):
        a = self.iw[0] * p

        for i in range(len(self.p_tdl)):
            a += self.iw[i + 1] * self.p_tdl[i]

        if len(self.p_tdl) > 0:
            self.p_tdl = np.roll(self.p_tdl, 1)
            self.p_tdl[0] = p

        return a

    def process(self, input_sequence):
        output = np.zeros(len(input_sequence))
        for i in range(len(input_sequence)):
            output[i] = self.step(input_sequence[i])
        return output
```

Initialize the TDL. If no values are provided, initialize with zeros.

Multiply weights time TDL elements to update one time step.

Update the TDL by rolling one element out and shifting in the new time point.

Process the entire input sequence.

The script below invokes the averaging network with the input from the example above and produces the correct network output.

```
# Input weights
iw = [0, 0.5, 0.5]

# Input sequence
p = [0, 1, 2, 3, 2, 1, 0, 0, 0]

# Define the network
net = averaging_network(iw)

# Run the input through the network
a = net.process(p)

print('Input sequence: ')
print(p)
print('Output sequence: ')
print(a)
```

```
Input sequence:
[0, 1, 2, 3, 2, 1, 0, 0, 0]
Output sequence:
[0.  0.  0.5 1.5 2.5 2.5 1.5 0.5 0. ]
```

Impulse response and convolution

In this section we introduce three concepts that are important when analyzing linear dynamic systems: the *impulse sequence*, the *impulse response* and the *convolution sum*. We introduced these ideas in the image processing context in Chapter 8, Convolution. There we saw that any linear, position-invariant transformation on an arbitrary image can be represented as the 2-D convolution sum of the impulse response of the transformation with the input image. The impulse in that case was an image with a single nonzero pixel. In this section we show that the response of any linear, *time-invariant* dynamic system to an arbitrary input sequence can be represented as the 1-D convolution sum of the impulse response of the dynamic system and the input sequence.

A *time-invariant system* is one for which the response to an input sequence that has been shifted in time is just a shifted version of the response to the unshifted input sequence.

In the image processing context, the unit impulse was the image with a single nonzero pixel with a value of 1. In the sequence processing context, the unit impulse is a sequence with one nonzero time point point, which occurs at $t = 0$. It is defined as

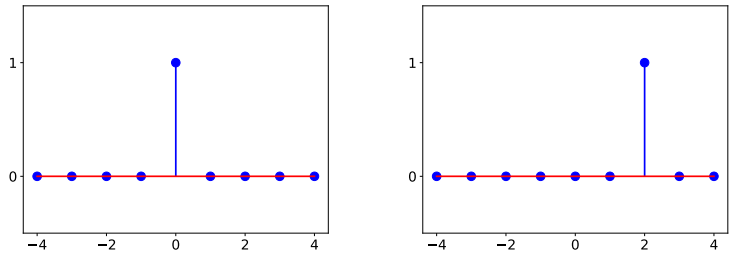
$$\delta(t) = \begin{cases} 1 & t = 0 \\ 0 & t \neq 0 \end{cases} \tag{11.5}$$

Table 11.2 tabulates the impulse sequence $\delta(t)$ and also shows a shifted version $\delta(t-2)$. A shifted version is nonzero when its argument is zero, in other words when $t - 2 = 0$, or $t = 2$.

t	-4	-3	-2	-1	0	1	2	3	4
$\delta(t)$	0	0	0	0	1	0	0	0	0
$\delta(t-2)$	0	0	0	0	0	0	1	0	0

Table 11.2: Impulse and shifted impulse

Figure 11.8 shows plots of the impulse sequence and its shifted version.



Impulse $\delta(t)$ Shifted Impulse $\delta(t-2)$

Figure 11.8: Impulse Sequences

The shifted impulse is useful in representing general sequences. For example, we can represent the input sequence in Figure 11.7 with the following formula:

$$p(t) = 1\delta(t-1) + 2\delta(t-2) + 3\delta(t-3) + 2\delta(t-4) + 1\delta(t-5) \tag{11.6}$$

To interpret Eq. 11.6, we want to keep in mind that we are combining five different sequences to produce $p(t)$. We can make this more concrete with Table 11.3, which shows the individual sequences and their sum.

We can represent any sequence $p(t)$ in a similar way, as follows:

$$p(t) = \sum_i p(i)\delta(t-i) \tag{11.7}$$

t	0	1	2	3	4	5	6	7	8
$1\delta(t-1)$	0	1	0	0	0	0	0	0	0
$2\delta(t-2)$	0	0	2	0	0	0	0	0	0
$3\delta(t-3)$	0	0	0	3	0	0	0	0	0
$2\delta(t-4)$	0	0	0	0	2	0	0	0	0
$1\delta(t-5)$	0	0	0	0	0	1	0	0	0
$p(t)$	0	1	2	3	4	1	0	0	0

Table 11.3: Summing shifted impulses

Now let's consider what happens when we apply an impulse input to a linear, time-invariant system. First, we generalize the sequence averaging system of Figure 11.6. If we allow the TDL to be of arbitrary size, we would have a system that satisfies Eq. 11.8.

$$a(t) = \sum_{i=1}^{n_\beta} \beta_i p(t-i) \tag{11.8}$$

where the coefficients β_i are fixed numbers (corresponding to weights in a neural network). This can be expanded to the *difference equation* in Eq. 11.9.

This is a *causal* system, since the output $a(t)$ does not depend on future values of the input sequence.

$$a(t) = \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta) \tag{11.9}$$

This system can be represented by the neural network in Figure 11.9, where the weight and bias are given in Eq. 11.10.

$$\mathbf{IW} = \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{n_\beta} \end{bmatrix}, b = 0 \tag{11.10}$$

Now apply an impulse sequence input to this system. The response is computed in Eq. 11.11 by substituting $\delta(t)$ for $p(t)$ in Eq. 11.8.

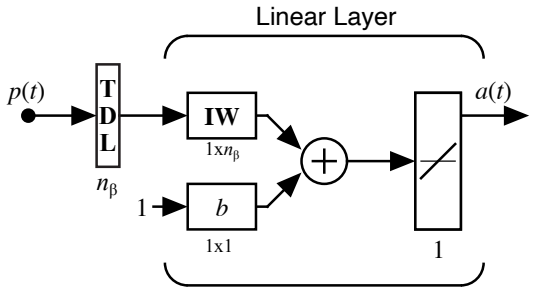


Figure 11.9: FIR Network

$$
\begin{aligned}
h(t) &= \sum_{i=1}^{n_\beta} \beta_i \delta(t-i) \\
h(1) &= \sum_{i=1}^{n_\beta} \beta_i \delta(1-i) = \beta_1 \\
h(2) &= \sum_{i=1}^{n_\beta} \beta_i \delta(2-i) = \beta_2 \\
&\vdots \\
h(n_\beta) &= \beta_{n_\beta}
\end{aligned} \tag{11.11}
$$

where $h(t)$ represents the system impulse response. We can write the impulse response in terms of the difference equation coefficients, as in Eq. 11.12.

$$
h(t) = \sum_{i=1}^{n_\beta} \beta_i \delta(t-i) = \begin{cases} \beta_t & t = 1, \cdots, n_\beta \\ 0 & \text{otherwise} \end{cases} \tag{11.12}
$$

The impulse response is illustrated in tabular form in Table 11.4.

t	0	1	2	3	4	$\cdots$	n_β	$n_\beta + 1$	$t > n_\beta$
$p(t) = \delta(t)$	1	0	0	0	0	0	0	0	0
$a(t) = h(t)$	0	β_1	β_2	β_3	β_4	$\cdots$	β_{n_β}	0	0

Table 11.4: Impulse Response

For this system the impulse response is the list of difference equation coefficients. Since $h(t) = 0$ for $t < 1$ and for $t > n_\beta$, the length of the impulse response sequence $h(t)$ is finite. That is why the system in Eq. 11.9 is called a *finite impulse response* (FIR) system.

Since the FIR coefficients in the difference equation are identical to the impulse response values, we can write

$$a(t) = \sum_{i=1}^{n_\beta} h(i)p(t-i) \tag{11.13}$$

Eq. 11.13 is called a convolution sum, and is often represented using the abbreviated notation in Eq. 11.14. The length of the impulse response can be a measure of the memory of the system. In this case, the system looks back at the last n_β values of the input sequence.

$$a(t) = h(t) * p(t) \tag{11.14}$$

We can show that the response of any linear, time-invariant system can be written as a convolution sum of the input sequence and the system impulse response. For example, consider a general input sequence $p(t)$, represented as impulses:

$$p(t) = \sum_i p(i)\delta(t-i) \tag{11.15}$$

and assume that a linear, time-invariant system $\mathcal{W}$ operates on that input:

$$a(t) = \mathcal{W}(p(t)) = \mathcal{W}\left(\sum_i p(i)\delta(t-i)\right) \tag{11.16}$$

Because of linearity and time invariance, the operation can be brought inside the summation, and the operation will be consistent at each time step. The response is a convolution of the impulse response and the input.

$$a(t) = \mathcal{W}(p(t)) = \sum_i p(i)\mathcal{W}(\delta(t-i)) = \sum_i p(i)h(t-i) = h(t) * p(t) \tag{11.17}$$

Infinite impulse response systems

In the FIR system of Eq. 11.9, $a(t)$ depends only on previous values of the input sequence $p(t)$. What if we were to modify the system so that $a(t)$ depends also on previous outputs? A simple system of that form is shown in Figure 11.10.

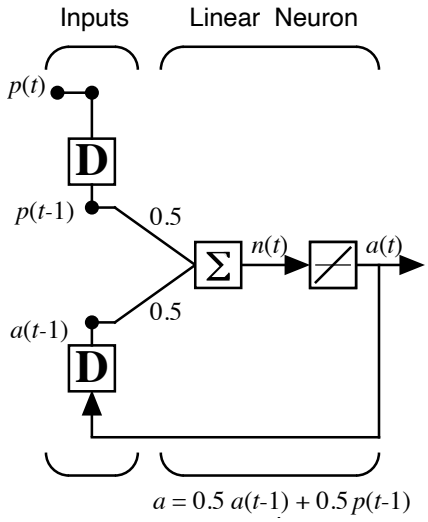


Figure 11.10: Sequence Smoothing Network

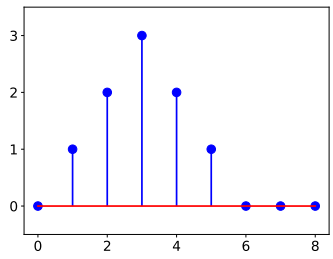
If we apply the input from Table 11.1 to this network, we get the response shown in Table 11.5. Here we have two TDLs – one for previous $p(t)$ and one for previous $a(t)$.

Table 11.5: Sequence Smoothing Network Response

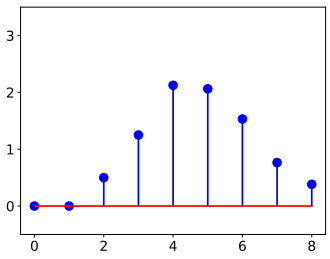
t	0	1	2	3	4	5	6	7	8
$p(t)$	0	1	2	3	2	1	0	0	0
$a(t)$	0	0	0.5	1.25	2.125	2.063	1.531	0.766	0.383
p_TDL	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}3\end{bmatrix}$	$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
a_TDL	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0.5\end{bmatrix}$	$\begin{bmatrix}1.25\end{bmatrix}$	$\begin{bmatrix}2.125\end{bmatrix}$	$\begin{bmatrix}2.063\end{bmatrix}$	$\begin{bmatrix}1.531\end{bmatrix}$	$\begin{bmatrix}0.766\end{bmatrix}$

A plot of the input and output sequences are shown in Figure 11.11. Note that, although we only have one delay of the input in the TDL, the output sequence extends more than one time step after the input sequence drops to zero. In fact, even if we extend the output sequence indefinitely it will not become zero. This network has infinite memory.

To illustrate that this network has infinite memory, the system impulse response is tabulated in Table 11.6.



Input Sequence $p(t)$



Output Sequence $a(t)$

Figure 11.11: Response of Sequence Smoothing Network

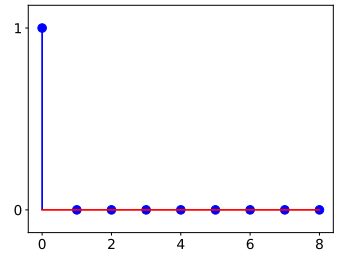
t	0	1	2	3	4	5	6	7	8
$p(t) = \delta(t)$	1	0	0	0	0	0	0	0	0
$a(t) = h(t)$	0	$\frac{1}{2}$	$\frac{1}{2^2}$	$\frac{1}{2^3}$	$\frac{1}{2^4}$	$\frac{1}{2^5}$	$\frac{1}{2^6}$	$\frac{1}{2^7}$	$\frac{1}{2^8}$
p_TDL	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
a_TDL	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^2}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^3}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^4}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^5}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^6}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{2^7}\end{bmatrix}$

Table 11.6: Impulse Response for Sequence Smoothing Network

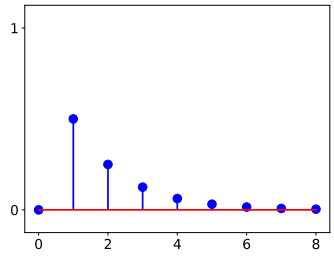
The cell for the initial TDL for $a(t)$ is highlighted in red, because the impulse response is defined as the response to the impulse sequence with all initial conditions set to zero.

Figure 11.12 displays the impulse response. The feedback connection produced an *infinite impulse response (IIR)*, therefore the memory of the network is infinite. However, we can see that it decays exponentially to zero. Although it has infinite memory, the effective memory is only six or seven steps, since the impulse response is less than 0.01 by step seven. As the feedback weight comes closer to 1, the effective memory length gets longer.

An IIR system can be approximated by an FIR system, if the TDL length for $p(t)$ is long enough, because their impulse responses can be made approximately the same.



Input Impulse $\delta(t)$



Impulse Response $h(t)$

Figure 11.12: Sequence Smoothing Network Impulse Response

To experiment with the sequence smoothing network, use the Deep Learning Demonstration Sequence Smoothing Response (**`dl11ssr`**).

A general IIR system can be written in the following form.

$$a(t) = \sum_{i=1}^{n_\alpha} \alpha_i a(t-i) + \sum_{i=1}^{n_\beta} \beta_i p(t-i) \tag{11.18}$$

If we expand the summation, we have the difference equation.

$$\begin{aligned} a(t) =& \alpha_1 a(t-1) + \alpha_2 a(t-2) + \ldots + \alpha_{n_\alpha} a(t-n_\alpha) \\ &+ \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta) \end{aligned} \tag{11.19}$$

This general IIR system can be implemented as a recurrent neural network, as shown in Figure 11.13, where the weights are

$$\mathbf{IW} = \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{n_\beta} \end{bmatrix} \tag{11.20}$$

$$\mathbf{LW} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n_\alpha} \end{bmatrix} \tag{11.21}$$

$$b = 0 \tag{11.22}$$

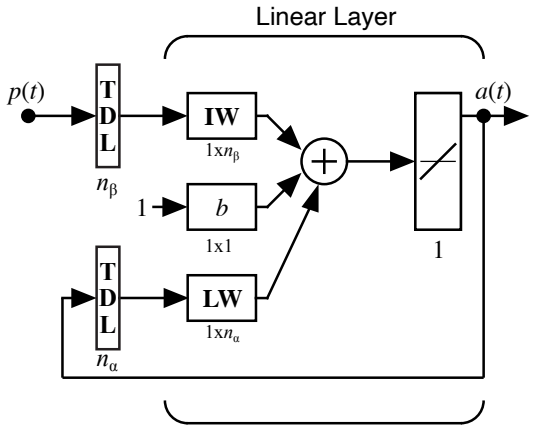


Figure 11.13: Infinite Impulse Response Network

When using IIR systems, which always have recurrent (feedback) connections, we face a problem that does not occur for FIR systems – *stability*. For example, consider the sequence smoothing network of Figure 11.10. If we were to change the feedback weight from 0.5 to 2, the impulse response would double at each time step, rather than decreasing by half.

One definition for stability of dynamic systems is that a bounded input will yield a bounded output.

To investigate stability for general linear dynamic systems, we need to introduce some additional notation. First, we define the delay operator D:

$$Da(t) = a(t-1) \tag{11.23}$$

We can then define the following:

$$\alpha(D) = 1 - \alpha_1 D - \alpha_2 D^2 - \ldots - \alpha_{n_\alpha} D^{n_\alpha} \tag{11.24}$$

$$\beta(D) = \beta_1 D + \beta_2 D^2 + \ldots + \beta_{n_\beta} D^{n_\beta} \tag{11.25}$$

This allows us to rewrite Eq. 11.19 as

$$a(t) = \frac{\beta(D)}{\alpha(D)} p(t) \tag{11.26}$$

It can be shown that the impulse response of the difference equation of Eq. 11.19 is determined from the roots of the denominator polynomial $\alpha(D)$. Setting $\alpha(D)$ to zero, we obtain the *characteristic equation* [Elaydi, 2005]:

$$\alpha(D) = \prod_{i-1}^{n_\alpha} (1 - \lambda_i D) = 0 \tag{11.27}$$

The impulse response for Eq. 11.19 will have the form

$$h(t) = A_1 (\lambda_1)^t + A_2 (\lambda_2)^t \cdots + A_n (\lambda_n)^t \tag{11.28}$$

if the characteristic roots λ_i (also called the system *poles*) are not repeated. In addition, we can say that the system will be stable if the poles are inside the unit circle (have magnitude less than one).

For example, consider the following difference equation.

$$a(t) = a(t-1) - 0.24a(t-2) + p(t-1) \tag{11.29}$$

Using the D operator, we can write:

$$a(t) = \frac{D}{1 - D + 0.24D^2} p(t) \tag{11.30}$$

The characteristic polynomial can be factored.

$$\alpha(D) = 1 - D + 0.24D^2 = (1 - 0.6D)(1 - 0.4D) \tag{11.31}$$

Because the poles $\lambda_1 = 0.6$ and $\lambda_2 = 0.4$ are inside the unit circle, the system is stable.

The impulse response of Eq. 11.29 is given in Eq. 11.32.

$$h(t) = 5(0.6)^t - 5(0.4)^t \tag{11.32}$$

It is plotted in Figure 11.14.

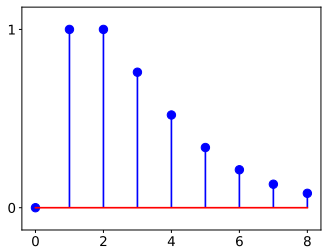


Figure 11.14: Impulse Response for Eq.11.29

To experiment with difference equations and impulse responses, use the Deep Learning Demonstration Impulse Response (**dl1ir**).

State space models

The difference equation models for dynamic systems that we have been considering are often called *input/output models* or *transfer function models*. There is another type of dynamic model - the *state space model*.

In the state space model, instead of a difference equation of order n, we have n first order difference equations:

$$\begin{aligned} q_i(t+1) =& \phi_{i,1}q_1(t) + \phi_{i,2}q_2(t) + \cdots + \phi_{i,n}q_n(t) \\ &+ \gamma_{i,1}p_1(t) + \gamma_{i,2}p_2(t) + \cdots + \gamma_{i,m}p_m(t) \qquad (11.33) \\ a_j(t) =& \psi_{j,1}q_1(t) + \psi_{j,2}q_2(t) + \cdots + \psi_{j,n}q_n(t) \qquad (11.34) \\ & \qquad (11.35) \end{aligned}$$

which can be written in matrix form as

$$\begin{aligned} \mathbf{q}(t+1) =& \mathbf{\Phi}\mathbf{q}(t) + \mathbf{\Gamma}\mathbf{p}(t) \qquad (11.36) \\ \mathbf{a}(t) =& \mathbf{\Psi}\mathbf{q}(t) \qquad (11.37) \end{aligned}$$

where the q_i are the *states* of the system. Knowing $\mathbf{q}(0)$ allows us to solve for all future states and outputs. The eigenvalues of $\mathbf{\Phi}$ are the system poles λ_i.

The state space model can be represented as a recurrent neural network, as shown in Figure 11.15, with the weights given in Eq. 11.38.

$$\mathbf{LW}^{1,1} = \mathbf{\Phi}, \quad \mathbf{IW}^{1,1} = \mathbf{\Gamma}, \quad \mathbf{LW}^{2,1} = \mathbf{\Psi}, \quad \mathbf{a}^1 = \mathbf{q}, \quad \mathbf{a}^2 = \mathbf{a} \qquad (11.38)$$

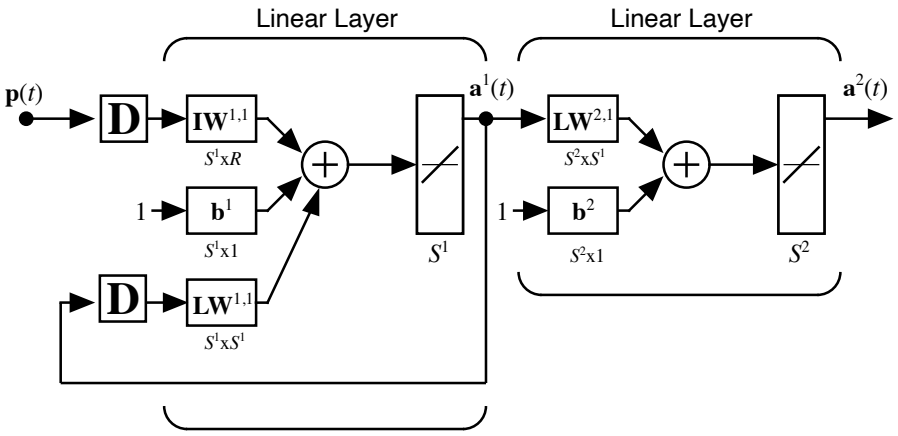


Figure 11.15: State Space Neural Network

For linear systems, it is always possible to convert from an input/output model to a state space model, although the conversion is not unique. In fact, for a given input/output model, there are an infinite number of equivalent state space models. It is worth demonstrating one method for doing the conversion, because it provides some insight into the meaning of the state.

Let's start with the general input/output difference equation of Eq. 11.19, which we repeat here for easy reference.

$$\begin{aligned} a(t) =& \alpha_1 a(t-1) + \alpha_2 a(t-2) + \ldots + \alpha_{n_\alpha} a(t-n_\alpha) \\ &+ \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta) \end{aligned} \tag{11.39}$$

This equation can be represented graphically by the simulation diagram in Figure 11.16. The output of the network is $a(t)$. Looking at the summation junction before the right-most delay, we see that two of the inputs are $\alpha_1 a(t)$ and $\beta_1 p(t)$. Since these two terms will be delayed once to become $a(t)$, they produce two terms on the right side of Eq. 11.39: $\alpha_1 a(t-1) + \beta_1 p(t-1)$. The additional terms appear at earlier delays, so that when they reach $a(t)$ they will be delayed the appropriate number of times.

We can use the simulation diagram to find a state space model. The state must contain the minimum number of variables so that if we know those variables at $t = 0$ and we know the input sequence we can solve for the output for all future time. From Figure 11.16,

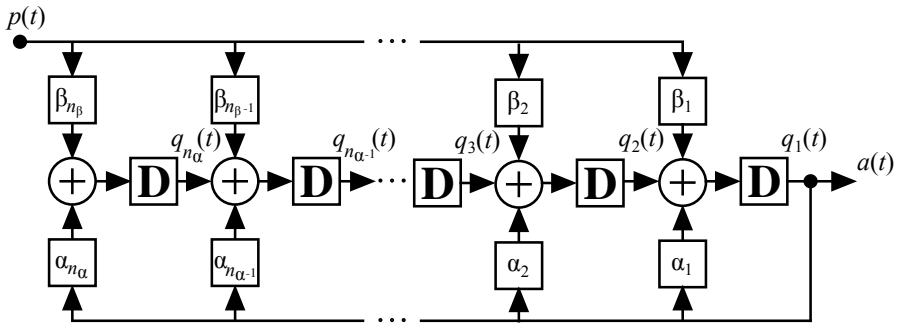


Figure 11.16: Simulation Diagram for IIR System

we can see that if we know the initial conditions on the delay blocks we can solve for $a(t)$ at all future times. This means that we can choose the outputs of the delays to be the states. Using the states as shown in Figure 11.16, the state equations become

$$\mathbf{q}(t+1) = \begin{bmatrix} \alpha_1 & 1 & 0 & \cdots & 0 \\ \alpha_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n_\alpha - 1} & 0 & 0 & \cdots & 1 \\ \alpha_{n_\alpha} & 0 & 0 & \cdots & 0 \end{bmatrix} \mathbf{q}(t) + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n_\beta - 1} \\ \beta_{n_\beta} \end{bmatrix} p(t) \qquad (11.40)$$

$$a(t) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \mathbf{q}(t) \qquad (11.41)$$

This is called the *observer canonical form* for the state equation. Other canonical forms can be found with different arrangements of the simulation diagram.

where each state equation comes from a summation junction in Figure 11.16.

To illustrate the process, consider again the second order system of Eq. 11.29, repeated below.

$$a(t) = a(t-1) - 0.24a(t-2) + p(t-1) \qquad (11.42)$$

The simulation diagram for that system is shown in Figure 11.17.

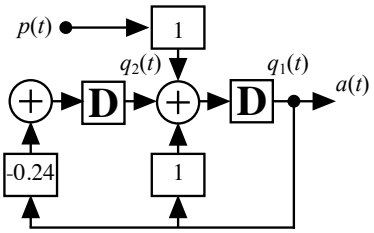


Figure 11.17: Simulation Diagram for Eq. 11.42

If we choose the outputs of the delays to be the state variables, we can write the state equations as follows:

$$q_1(t+1) = 1q_1(t) + 1q_2(t) + 1p(t) \tag{11.43}$$
$$q_2(t+1) = -0.24q_1(t) + 0q_2(t) + 0p(t) \tag{11.44}$$
$$a(t) = 1q_1(t) + 0q_2(t) \tag{11.45}$$

and they can be put in matrix form.

$$\mathbf{q}(t+1) = \begin{bmatrix} 1 & 1 \\ -0.24 & 0 \end{bmatrix} \mathbf{q}(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} p(t) \tag{11.46}$$
$$a(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{q}(t) \tag{11.47}$$

Epilogue

Sequence processing is one of the most important applications of deep learning. Language translation, sentiment analysis, chatbots, financial prediction, weather forecasting, protein structure prediction, drug discovery, music generation, speech synthesis, and video captioning are just a few of the ways that deep learning has been used for sequence processing.

This chapter is just the beginning of our investigation into sequence processing. The linear methods introduced in this chapter have introduced many of the key concepts that we will build on over the next few chapters. In those chapters we will again see tapped delay lines, finite memory networks, infinite memory networks, input/output models and state space models.

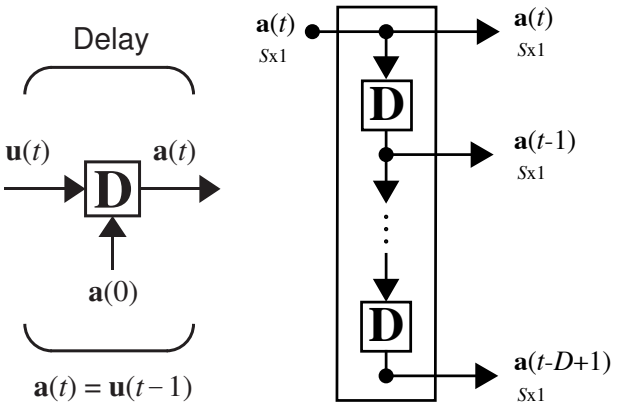
Further Reading

[Elaydi, 2005] Saber N Elaydi. *An Introduction to Difference Equations (3rd Edition)*. New York, NY: Springer New York: Imprint: Springer,, 2005

Chapter 2 of this text, Linear Difference Equations, provides a comprehensive treatment of the theory connecting characteristic polynomials to difference equation solutions, including proofs of the fundamental theorems and extensive examples. Chapter 3, Systems of Linear Difference Equations, covers state space models and how to convert from nth order difference equations to state space models.

Summary of Results

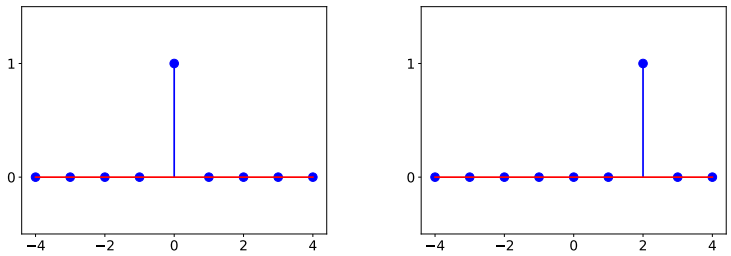
Delay and Tapped Delay Line (TDL)



(a) Delay (b) Tapped Delay Line

Impulse Function

$$\delta(t) = \begin{cases} 1 & t = 0 \\ 0 & t \neq 0 \end{cases}$$

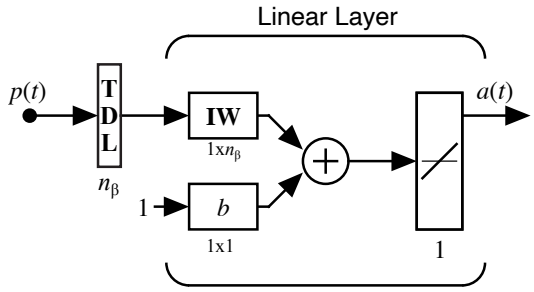


Impulse $\delta(t)$ Shifted Impulse $\delta(t-2)$

Finite Impulse Response (FIR) System

$$a(t) = \sum_{i=1}^{n_\beta} \beta_i p(t-i)$$

$$a(t) = \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta)$$



$$\mathbf{IW} = \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{n_\beta} \end{bmatrix}, b = 0$$

$$h(t) = \sum_{i=1}^{n_\beta} \beta_i \delta(t-i) = \begin{cases} \beta_t & t = 1, \cdots, n_\beta \\ 0 & \text{otherwise} \end{cases}$$

Convolution Sum

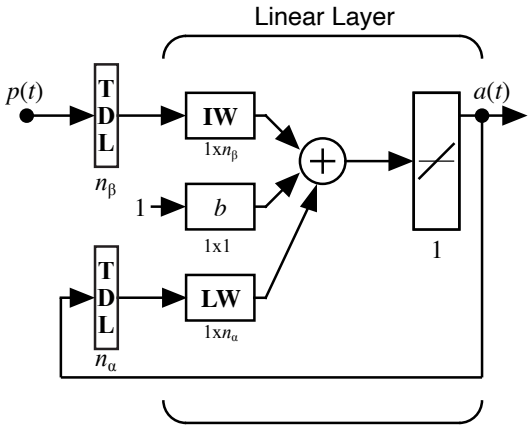
$$a(t) = \mathcal{W}(p(t)) = \sum_i p(i)\mathcal{W}(\delta(t-i)) = \sum_i p(i)h(t-i) = h(t) * p(t)$$

Infinite Impulse Response (FIR) System

$$a(t) = \sum_{i=1}^{n_\alpha} \alpha_i a(t-i) + \sum_{i=1}^{n_\beta} \beta_i p(t-i)$$

$$\begin{aligned} a(t) =& \alpha_1 a(t-1) + \alpha_2 a(t-2) + \ldots + \alpha_{n_\alpha} a(t-n_\alpha) \\ &+ \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta) \end{aligned}$$

$$\begin{aligned} \mathbf{IW} &= \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{n_\beta} \end{bmatrix} \\ \mathbf{LW} &= \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n_\alpha} \end{bmatrix} \\ b &= 0 \end{aligned}$$



Stability

$$Da(t) = a(t-1)$$

$$\alpha(D) = 1 - \alpha_1 D - \alpha_2 D^2 - \ldots - \alpha_{n_\alpha} D^{n_\alpha}$$
$$\beta(D) = \beta_1 D + \beta_2 D^2 + \ldots + \beta_{n_\beta} D^{n_\beta}$$

$$a(t) = \frac{\beta(D)}{\alpha(D)} p(t)$$

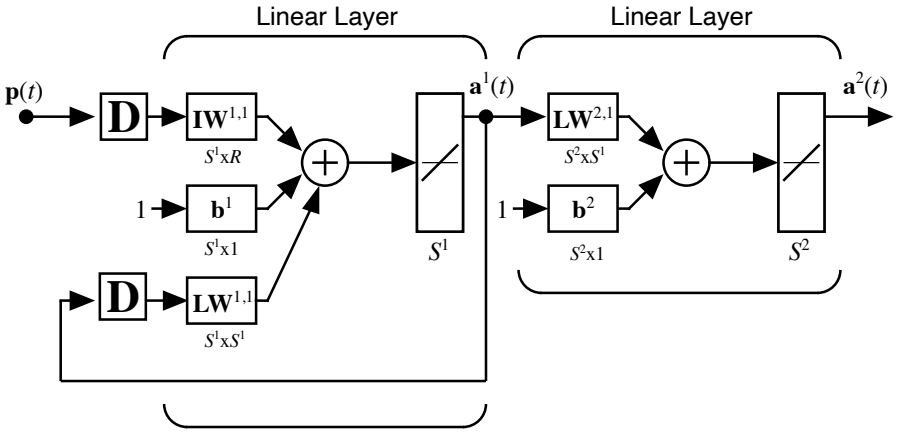
$$\alpha(D) = \prod_{i-1}^{n_\alpha} (1 - \lambda_i D) = 0$$

$$h(t) = A_1 (\lambda_1)^t + A_2 (\lambda_2)^t \cdots + A_n (\lambda_n)^t$$

The system is stable if $|\lambda_i| < 1$ for all i.

State Space Models

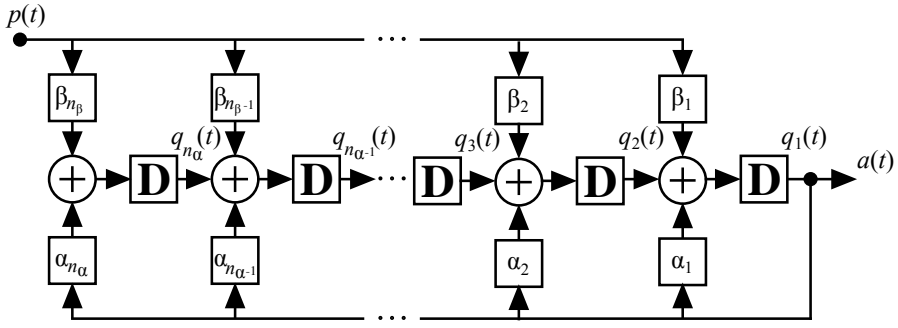
$$\mathbf{q}(t+1) = \mathbf{\Phi q}(t) + \mathbf{\Gamma p}(t)$$
$$\mathbf{a}(t) = \mathbf{\Psi q}(t)$$



$$\mathbf{LW^{1,1}} = \mathbf{\Phi}, \quad \mathbf{IW^{1,1}} = \mathbf{\Gamma}, \quad \mathbf{LW^{2,1}} = \mathbf{\Psi}, \quad \mathbf{a}^1 = \mathbf{q}, \quad \mathbf{a}^2 = \mathbf{a}$$

Converting from Input/Output to State Space

$$\begin{aligned} a(t) =& \alpha_1 a(t-1) + \alpha_2 a(t-2) + \ldots + \alpha_{n_\alpha} a(t-n_\alpha) \\ &+ \beta_1 p(t-1) + \beta_2 p(t-2) + \ldots + \beta_{n_\beta} p(t-n_\beta) \end{aligned}$$



$$\mathbf{q}(t+1) = \begin{bmatrix} \alpha_1 & 1 & 0 & \cdots & 0 \\ \alpha_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n_\alpha - 1} & 0 & 0 & \cdots & 1 \\ \alpha_{n_\alpha} & 0 & 0 & \cdots & 0 \end{bmatrix} \mathbf{q}(t) + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n_\beta - 1} \\ \beta_{n_\beta} \end{bmatrix} p(t)$$

$$a(t) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \mathbf{q}(t)$$

Solved Problems

P11.1 **Consider the following difference equation.**

$$a(t) = 3a(t-1) - 2a(t-2) + 2p(t-1) - 2p(t-2)$$

i. **Represent the system as a neural network, using our notation.**

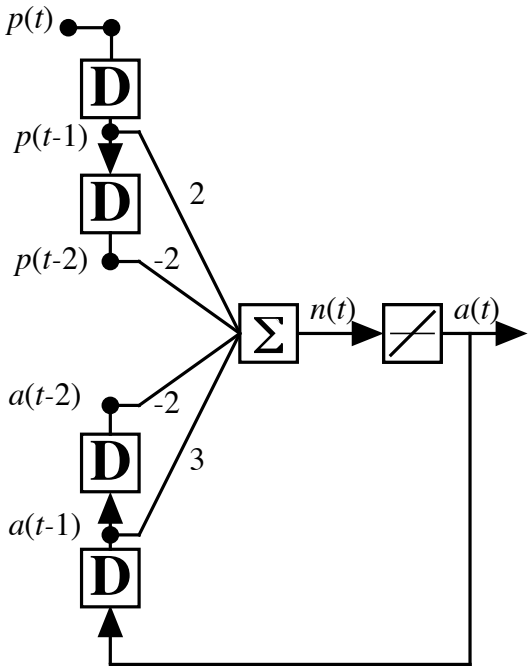
ii. **Find the impulse response by iterating the difference equation by hand up to $t = 5$, and create a table like Table 11.6.**

iii. **Plot the impulse response**

iv. **Find the system poles, and determine if the system is stable.**

v. **Knowing that the impulse response will have the form of Eq. 11.28, if the poles are not repeated, find a closed form representation of the impulse response.**

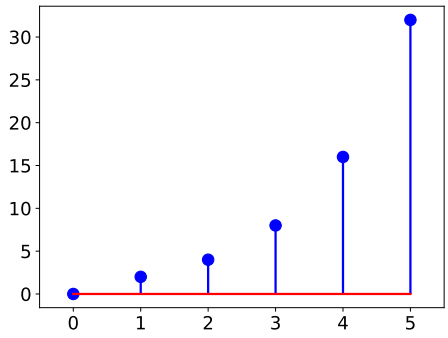
i. Below is the network diagram.



ii. The following table shows the steps of computing the impulse response by iterating the difference equation.

t	0	1	2	3	4	5
$p(t)$	1	0	0	0	0	0
$a(t)$	0	2	4	8	16	32
p_TDL	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$
a_TDL	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}2\\0\end{bmatrix}$	$\begin{bmatrix}4\\2\end{bmatrix}$	$\begin{bmatrix}8\\4\end{bmatrix}$	$\begin{bmatrix}16\\8\end{bmatrix}$

iii. Here is the plot of the impulse response.



iv. We can represent the difference equation as

$$a(t) = \frac{\beta(D)}{\alpha(D)}$$

where

$$\alpha(D) = 1 - 3D + 2D^2$$
$$\beta(D) = 2D - 2D^2$$

We factor the denominator polynomial $\alpha(D)$ to find the system poles.

$$\alpha(D) = 1 - 3D + 2D^2 = (1 - \lambda_1 D)(1 - \lambda_2 D)$$

The λ_i are the inverse of the roots of the $a(D)$ polynomial, so we can find the roots of that polynomial first. From the quadratic formula we have

$$\frac{3 \pm \sqrt{9 - 4 \times 2}}{2 \times 2} = \frac{3 \pm 1}{4} = 1, \frac{1}{2}$$

So the poles are $\lambda_1 = 1$ and $\lambda_2 = 2$. Since the poles are not inside the unit circle, the system is not stable.

v. We know from Eq. 11.28 that, since the poles are not repeated, the impulse response should have the form

$$h(t) = A_1 (1)^t + A_2 (2)^t$$

To find the coefficients A_1 and A_2, we just need two values for $h(t)$. We found $a(1)$ and $a(2)$ in part ii., so we can use those two values.

$$\begin{aligned} h(1) = 2 &= A_1 (1)^1 + A_2 (2)^1 = 1A_1 + 2A_2 \\ h(2) = 4 &= A_1 (1)^2 + A_2 (2)^2 = 1A_1 + 4A_2 \end{aligned}$$

We can write this in matrix form.

$$\begin{bmatrix} 1 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Therefore, $A_1 = 0$, $A_2 = 1$ and the impulse response is

$$h(t) = (2)^t, \, t > 0$$

P11.2 **Consider again the difference equation in Solved Problem P.11.1. Assume that the following input is applied to the system.**

$$p(t) = 1\delta(t-1) + 1\delta(t-2)$$

i. **Find the response of the system for the first five time steps by iterating the difference equation.**

ii. Find the response of the system for the first five time steps by performing a convolution sum between the input and the impulse response.

i. This table shows the steps for iterating the difference equation.

t	0	1	2	3	4	5
$p(t)$	1	1	0	0	0	0
$a(t)$	0	2	6	12	24	48
p_TDL	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$
a_TDL	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}2\\0\end{bmatrix}$	$\begin{bmatrix}6\\2\end{bmatrix}$	$\begin{bmatrix}12\\6\end{bmatrix}$	$\begin{bmatrix}24\\12\end{bmatrix}$

ii. To compute the output using the convolution summation, we use Eq. 11.17.

$$a(t) = \sum_i p(i)h(t-i)$$

Since $p(i) = 0$ for $i < 0$, the lower limit on the sum will be 0. Since $h(j) = 0$ for $j < 1$, the upper limit on the sum will be $i = t - 1$. From the previous problem we know that $h(t) = 2^t$ over the range where it is not zero. Therefore the convolution sum reduces to

$$a(t) = \sum_{i=0}^{t-1} p(i)2^{(t-i)}$$

Also, for this case, $p(t) = 0$ for $t > 1$, so the upper limit of the sum is 1, if $t - 1 > 1$:

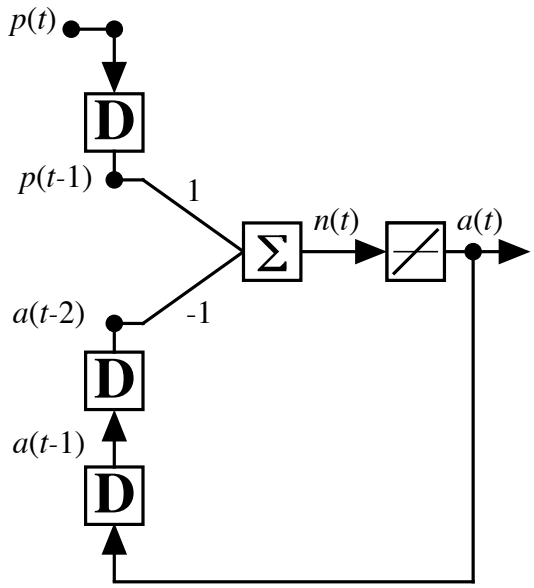
$$a(t) = \sum_{i=0}^{\min(t-1,1)} p(i)2^{(t-i)}$$

Plugging in values for $p(t)$, we obtain

$$\begin{aligned}
a(1) &= p(0)2^1 = (1)(2) = 2 \\
a(2) &= p(0)2^2 + p(1)2^1 = (1)(4) + (1)(2) = 6 \\
a(3) &= p(0)2^3 + p(1)2^2 = (1)(8) + (1)(4) = 12 \\
a(4) &= p(0)2^4 + p(1)2^3 = (1)(16) + (1)(8) = 24 \\
a(5) &= p(0)2^5 + p(1)2^4 = (1)(32) + (1)(16) = 48
\end{aligned}$$

which matches the results from part i.

P11.3 Consider the network shown below.



i. Find the corresponding difference equation.

ii. Find the system poles.

iii. Find the system system impulse response in closed form.

i. The difference equation is

$$a(t) = -a(t-2) + p(t-1)$$

i. To find the poles, we set the $\alpha(D)$ polynomial to zero, and then find the inverse roots

$$\alpha(D) = 1 + D^2 = 0$$

Since we want the inverse roots, we could introduce $E = D^{-1}$

$$1 + D^2 = 1 + E^{-2} = 0$$
$$E^2 + 1 = 0$$

Where we multiplied both sides by E^2 to get the last equation. We can then find the roots of the polynomial in E to find the poles. From the quadratic equation we have

$$E = \frac{0 \pm \sqrt{0-4}}{2} = \frac{\pm 2j}{2} = \pm j$$

The poles are therefore $\lambda_1 = j$ and $\lambda_2 = -j$, and the impulse response has the form

$$h(t) = A_1 (j)^t + A_2 (-j)^t$$

We need two values for $h(t)$ to solve for A_1 and A_2. Iterating the difference equation twice, we get

$$a(1) = -a(-1) + p(0) = -0 + 1 = 1$$
$$a(2) = -a(0) + p(1) = -0 + 0 = 0$$

Plugging these values in for $h(1)$ and $h(2)$ in the earlier equation, we have

$$h(1) = 1 = A_1(j) + A_2(-j)$$
$$h(2) = 0 = A_1(j)^2 + A_2(-j)^2 = -A_1 + -A_2$$

We can put this in matrix form:

$$\begin{bmatrix} j & -j \\ -1 & -1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} j & -j \\ -1 & -1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2j} & -\frac{1}{2} \\ -\frac{1}{2j} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2j} \\ -\frac{1}{2j} \end{bmatrix}$$

Therefore $A_1 = \frac{1}{2j}$ and $A_2 = -\frac{1}{2j}$. We can then write $h(t)$ as

$$h(t) = \frac{1}{2j}(j)^t - \frac{1}{2j}(-j)^t = \frac{(j)^t - (-j)^t}{2j}$$

There is a trigonometric identity

$$\sin(\theta) = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

Since $j = e^{j\frac{\pi}{2}}$ and $-j = e^{-j\frac{\pi}{2}}$, we can write $h(t)$ as

$$h(t) = \frac{e^{j\frac{\pi}{2}t} - e^{-j\frac{\pi}{2}t}}{2j} = \sin\left(\frac{\pi}{2}t\right)$$

Note: Compare this result to the impulse response of the network in Figure 11.5. That system also has complex poles. Complex poles cause oscillation, while real poles produce exponential responses. When complex poles are on the unit circle, the oscillation does not decay, as in this problem. If the poles are inside the unit circle, the oscillation will decay. If the pole are outside the unit circle, the oscillation will explode.

P11.4 Is there a convolution sum formulation for systems in state space form?

The state space form is

$$\begin{aligned}\mathbf{q}(t+1) =& \mathbf{\Phi}\mathbf{q}(t) + \mathbf{\Gamma}\mathbf{p}(t)\\ \mathbf{a}(t) =& \mathbf{\Psi}\mathbf{q}(t)\end{aligned}$$

If we assume that the system starts with initial state $\mathbf{q}(0)$, we can iterate the state equation as follows.

$$\begin{aligned}\mathbf{q}(1) =& \mathbf{\Phi}\mathbf{q}(0) + \mathbf{\Gamma}\mathbf{p}(0)\\ \mathbf{q}(2) =& \mathbf{\Phi}\mathbf{q}(1) + \mathbf{\Gamma}\mathbf{p}(1) = \mathbf{\Phi}^2\mathbf{q}(0) + \mathbf{\Phi}\mathbf{\Gamma}\mathbf{p}(0) + \mathbf{\Gamma}\mathbf{p}(1)\\ \mathbf{q}(3) =& \mathbf{\Phi}\mathbf{q}(2) + \mathbf{\Gamma}\mathbf{p}(2) = \mathbf{\Phi}^3\mathbf{q}(0) + \mathbf{\Phi}^2\mathbf{\Gamma}\mathbf{p}(0) + \mathbf{\Phi}\mathbf{\Gamma}\mathbf{p}(1) + \mathbf{\Gamma}\mathbf{p}(2)\end{aligned}$$

We can see that a pattern is emerging. We can write a general expression at time point t. The second term is a convolution sum.

$$\mathbf{q}(t) = \mathbf{\Phi}^t\mathbf{q}(0) + \sum_{i=1}^{t} \mathbf{\Phi}^{t-i}\mathbf{\Gamma}\mathbf{p}(i-1)$$

Exercises

E11.1 For the following difference equations, a) represent the system as a neural network (using our notation) and b) find and plot (for $0 < t < 8$) the response of the system to the following input sequence.

$$p(t) = \delta(t-2) + \delta(t-3) + \delta(t-4)$$

i. $a(t) = \frac{1}{3}p(t-1) + \frac{1}{3}p(t-2) + \frac{1}{3}p(t-3)$

ii. $a(t) = p(t-1) - p(t-2)$

iii. $a(t) = p(t-1) - 0.5p(t-2) - 0.25p(t-3) - 0.125p(t-4)$

E11.2 For the following difference equations, a) represent the system as a neural network, using our notation, b) find the impulse response by iterating the difference equation by hand up to $t = 5$, c) plot the impulse response, d) find the system poles and check stability, and e) find a closed form representation of the impulse response.

i. $a(t) = -5a(t-1) - 6a(t-2) + p(t-1)$

ii. $a(t) = 1.62a(t-1) - a(t-2) + 0.6p(t-1)$

iii. $a(t) = -a(t-1) - 0.21a(t-2) + p(t-1)$

iv. $a(t) = 0.707a(t-1) - 0.25a(t-2) + 2p(t-1)$

v. $a(t) = -2a(t-1) - 3a(t-2) - 2a(t-3) + p(t-1) + 4p(t-2)$

E11.3 For the systems in Exercise E.11.2, find the responses (for $0 < t < 4$) to the input $p(t)$ in Exercise E.11.1 using the convolution sums with the impulse responses.

E11.4 For the system in Exercise E.11.2 iii., find an FIR system that would have approximately the same response to any input sequence. Would this be possible for the system in Exercise E.11.2 i.?

E11.5 If $p(t-1)$ and $p(t-2)$ are changed to $p(t)$ and $p(t-1)$ in Exercise E.11.2, how will the impulse response change? Explain.

E11.6 If every $a(t-i)$ and $p(t-i)$ in Exercise E.11.2 was changed to $a(t+i)$ and $p(t+i)$, how would the resulting systems be implemented? How would the response to input sequences change? Explain.

E11.7 If a dynamic system has the impulse response below, find the corresponding difference equation representation of the system.

$$a(t) = (0.5)^t + (0.75)^t, \text{ for } t > 0$$

E11.8 If a system has complex poles $r\exp(\pm j\theta)$ (in polar form), then, based on Eq. 11.28, the following terms will appear in the impulse response:

$$A_1\left[r\exp(j\theta)\right]^t + A_2\left[r\exp(-j\theta)\right]^t = A_1 r^t \exp(j\theta t) + A_2 r^t \exp(-j\theta t)$$

i. Since this expression must be real, show that A_2 must be a complex conjugate of A_1.

ii. Show that the expression above can then be rewritten as follows:

$$2\left|A_1\right| r^t \cos(\theta t + \angle A_1)$$

iii. Using this concept, find the impulse response for the network in Figure 11.5. (The impulse response is shown in Figure 11.2.)

E11.9 Consider again the difference equations in Exercise E.11.2.

i. Find the simulation diagrams.

ii. Find the state space representations.

iii. Verify that the eigenvalues of $\mathbf{\Phi}$ match the system poles you found in Exercise E.11.2, part d).

E11.10 Perform Lab 1, which can be found at `https://github.com/NNDesignDeepLearning/NNDesignDeepLearning/tree/master/NNDesignDeepLearning/11.LinearSequenceProcessingChapter/Code/Labs`