浙江大學
ZHEJIANG UNIVERSITY

智能控制技术
实验报告

实验名称 神经网络辨识作业

姓 名 _____

学 号 _____

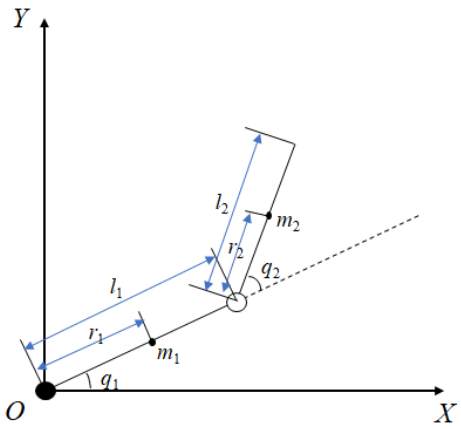
提交日期 December 19, 2024

指导老师 刘山

1 问题分析

1.1 问题重述

如图所示二自由度机械臂模型（平面俯视图），q_1 和 q_2 表示机械臂的两个关节角大小。



图中，m_i、l_i、$r_i(i=1,2)$ 分别为两连杆的质量、连杆长度和质心到相应关节的距离。两个连杆的转动惯量分别为 I_1 和 I_2。该机械臂动力学方程表示为：

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{1}$$

$M(q)$ 为惯性矩阵，$C(q,\dot{q})$ 为科氏力和向心力的结合矩阵，$G(q)$ 为重力势能矩阵。τ 为驱动力矩的向量。式 (1) 可写为如下方式：

$$m_{11}\ddot{q}_1 + m_{12}\ddot{q}_2 + c_{11}\dot{q}_1 + c_{12}\dot{q}_2 + g_1 = \tau_1 \tag{2}$$

$$m_{21}\ddot{q}_1 + m_{22}\ddot{q}_2 + c_{21}\dot{q}_1 + c_{22}\dot{q}_2 + g_2 = \tau_2 \tag{3}$$

τ_1、τ_2 分别为关节 1 和关节 2 的驱动力矩。

定义以下参数：

$$\begin{aligned}
h_1 &= m_1 r_1^2 + m_2 l_2^2 + I_1 \\
h_2 &= m_2 r_2^2 + I_2 \\
h_3 &= m_2 l_1 r_2 \\
h_4 &= m_1 r_1 + m_2 l_1 \\
h_5 &= m_2 r_2
\end{aligned}$$

则式 (2) 和式 (3) 中的参数可按如下计算：

$$
\begin{aligned}
m_{11} &= h_1 + h_2 + 2h_3\cos(q_2) \\
m_{12} &= m_{21} = h_2 + h_3\cos(q_2) \\
m_{22} &= h_2 \\
c_{11} &= -h_3\sin(q_2)\dot{q}_2 \\
c_{12} &= -h_3\sin(q_2)(\dot{q}_1 + \dot{q}_2) \\
c_{21} &= h_3\sin(q_2)\dot{q}_1 \\
c_{22} &= 0 \\
g_1 &= h_4 g\cos(q_1) + h_5 g\cos(q_1 + q_2) \\
g_2 &= h_5 g\cos(q_1 + q_2)
\end{aligned}
$$

式中，g 为重力加速度 $9.8\,\mathrm{m/s^2}$。

假定系统参数如下表所示：

参数	值
h_1	0.0308
h_2	0.0106
h_3	0.0095
h_4	0.2086
h_5	0.0631

Table 1: 系统参数表

请设计神经网络辨识方案，对该系统进行辨识（系统输入为 τ_1、τ_2，输出为 q_1、q_2）。

参考步骤：

(1). 利用已知系统得到辨识所需的输入输出数据；

(2). 通过步骤 1 得到的数据来训练神经网络；

(3). 对比原系统与神经网络辨识得到的系统是否一致。（给两个系统同样的输入，观察输出是否相同）

（可以利用 Matlab 中的相关工具箱进行仿真）

1.2 问题分析

1.2.1 模型分析

在本问题中，给定了一个二自由度机械臂的动力学模型，其中包含了关节位置、速度、加速度的相关变量，以及该机械臂的驱动力矩和外部影响的因素。目标是设计一个神经网络辨识方案，通过已有的动力学模型训练神经网络，使得该网络能够通过输入的驱动力矩（τ_1, τ_2）来预测系统的状态（q_1, q_2）。

系统的动力学方程如公式 (1) 所示为了将动力学方程转化为状态空间模型，需要将加速度（$\ddot{q_1}, \ddot{q_2}$）表示为状态变量的函数。通过选定的状态变量（位置和速度），我们得到了对应的状态方程。

状态变量选择如下：

$$x = \begin{bmatrix} q_1 \\ \dot{q_1} \\ q_2 \\ \dot{q_2} \end{bmatrix} \tag{4}$$

其中，$q_1, \dot{q_1}, q_2, \dot{q_2}$ 分别表示机械臂两个关节的角度和角速度。通过求解该状态方程，可以得到加速度的表达式，并进一步获得系统的状态空间模型。

根据题目给出的系统参数，如连杆质量、长度、转动惯量等，通过代入系统的动力学方程，可以计算出惯性矩阵。在模型建立之后，利用已知的系统参数进行仿真，收集系统的输入（τ_1, τ_2）和输出（q_1, q_2）。通过 Simulink 仿真工具，采集到系统的输入输出数据后，这些数据将作为神经网络的训练数据。

1.2.2 神经网络搭建

对于神经网络用于系统辨识方面，常利用多层静态网络进行系统辨识，要求预先给定系统的阶，即必须给出定阶的差分方程，典型的是 NARMA 模型 (非线性自回归滑动平均模型)。一般可采用两种辨识模型

(1). 并联模型：(动态反馈前向网络)

$$\hat{y}_p(k+1) = f\left[\hat{y}_p(k), \hat{y}_p(k-1), \ldots, \hat{y}_p(k-n+1); u(k), u(k-1), \ldots, u(k-m+1)\right] \tag{5}$$

(2). 串-并联模型：（静态前向网络）

$$\hat{y}_p(k+1) = f\left[y_p(k), y_p(k-1), \ldots, y_p(k-n+1); u(k), u(k-1), \ldots, u(k-m+1)\right] \tag{6}$$

根据上面两种辨识模型，来确定模型的输入输出参数的数据，进行训练后，将实际的系统输出，与神经网络系统的输出做比较，观察其偏差。

1.2.3 提高模型检测方法

要优化神经网络的效果，可以采取如下几种方法：

(1). 增加延时个数: 延时（Lag）指的是输入序列中引入的历史时间点的数据。增加延时个数可以帮助网络更好地捕捉时间序列数据中的长期依赖关系。常见的方式是增加输入特征的数量，使模型能够获得更多的过去信息。

(2). 提高数据量: 可以通过增加白噪声的种子值得到的输入输出数据作为训练数据集，来增大数据量以及提高模型的泛化性，

(3). 增加隐藏神经元个数：隐藏层神经元的数量直接影响神经网络的学习能力。增加隐藏层神经元个数能够提高网络的容量，使其能够学习更复杂的模式，但是不能过多。

2 问题求解

2.1 状态空间模型

2.1.1 模型求解

根据题目给出的机械臂的动力学方程，我们可以推导出系统的状态方程。系统的动力学方程如下：

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{7}$$

$M(q)$ 为惯性矩阵，$C(q,\dot{q})$ 为科氏力和向心力的结合矩阵，$G(q)$ 为重力势能矩阵。τ 为驱动力矩的向量。

为了将系统转化为状态方程，我们需要定义状态变量。常见的做法是将位置和速度作为状态变量。令：

状态变量	**物理意义**
x_1	q_1 表示关节 1 的位置（角度）
x_2	$\dot{q}_1$ 表示关节 1 的速度
x_3	q_2 表示关节 2 的位置（角度）
x_4	$\dot{q}_2$ 表示关节 2 的速度

因此，状态变量为：

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ \dot{q}_1 \\ q_2 \\ \dot{q}_2 \end{bmatrix} \tag{8}$$

而状态方程就是描述这些状态变量随时间的变化。我们可以将原始的动力学方程进行变换，得到状态方程。其中原始的动力学方程为可以将其分解为两部分：

$$\ddot{q} = M^{-1}(q)\left(\tau - C(q,\dot{q})\dot{q} - G(q)\right) \tag{9}$$

为了得到状态方程，我们需要将加速度 $\ddot{q}$ 表达成状态变量的形式。因此，根据状态变量的定义，状态方程可以写成：

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ \ddot{q_1} \\ x_4 \\ \ddot{q_2} \end{bmatrix} \tag{10}$$

根据之前的动力学方程，我们可以得到加速度 $\ddot{q}_1$ 和 $\ddot{q}_2$ 的表达式：

$$\begin{cases} \ddot{q_1} = [M(q)]^{-1} \left(\tau_1 - C(q,\dot{q})\dot{q} - G(q)\right)_1 \\ \ddot{q_2} = [M(q)]^{-1} \left(\tau_2 - C(q,\dot{q})\dot{q} - G(q)\right)_2 \end{cases} \tag{11}$$

其中，$[M(q)]^{-1}$ 是惯性矩阵的逆矩阵，表示系统的质量分布特性。为了简化，下面我们将这些表达式代入状态方程。

综上所述，状态方程的形式为：

$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = \ddot{q_1} = [M(q)]^{-1} \left(\tau_1 - C(q,\dot{q})\dot{q} - G(q)\right)_1 \\ \dot{x_3} = x_4 \\ \dot{x_4} = \ddot{q_2} = [M(q)]^{-1} \left(\tau_2 - C(q,\dot{q})\dot{q} - G(q)\right)_2 \end{cases} \tag{12}$$

因此，完整的状态方程系统为：

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ [M(q)]^{-1} \left(\tau_1 - C(q,\dot{q})\dot{q} - G(q)\right)_1 \\ x_4 \\ [M(q)]^{-1} \left(\tau_2 - C(q,\dot{q})\dot{q} - G(q)\right)_2 \end{bmatrix} \tag{13}$$

其中，τ_1 和 τ_2 是系统的输入，q_1, q_2 是输出（关节角度），$\dot{q_1}$ 和 $\dot{q_2}$ 是速度，$\ddot{q_1}$ 和 $\ddot{q_2}$ 是加速度。

2.1.2 S-Function 函数建立

在上面已经将状态空间模型求解出来之后，可以建立 S-Function 模型来实现系统的建模, 具体代码如下 (**详细注释见文件代码 system1.m**)

类型	变量名	说明
输入	τ_1	关节 1 的驱动力矩
输入	τ_2	关节 2 的驱动力矩
输出	q_1	关节 1 的位置（角度）
输出	q_2	关节 2 的位置（角度）
状态变量	$x_1 = q_1$	关节 1 的位置（角度）
状态变量	$x_2 = \dot{q}_1$	关节 1 的速度
状态变量	$x_3 = q_2$	关节 2 的位置（角度）
状态变量	$x_4 = \dot{q}_2$	关节 2 的速度

Table 2: system1 的输入与输出说明

Listing 1: SFunction 代码

```
function [sys,x0,str,ts,simStateCompliance] = demo(t,x,u,flag)
    switch flag
        case 0
            % 初始化
            [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
        case 1
            % 计算连续状态导数
            sys = mdlDerivatives(t,x,u);
        case 2
            % 更新离散状态（如果有）
            sys = mdlUpdate(t,x,u);
        case 3
            % 计算输出
            sys = mdlOutputs(t,x,u);
        case 4
            % 下一个采样时间（用于可变步长仿真）
            sys = mdlGetTimeOfNextVarHit(t,x,u);
        case 9
            % 仿真结束前的清理工作
            sys = mdlTerminate(t,x,u);
        otherwise
            % 未处理的flag错误
            DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag))
                ;
```

```
24      end
25  % 主函数结束
26  %% %下面是各个子函数，即各个回调过程
27  function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
28      sizes = simsizes;           %生成sizes数据结构，信息被包含在其中
29      sizes.NumContStates = 4;    %连续状态数，缺省为0
30      sizes.NumDiscStates = 0;    %离散状态数，缺省为0
31      sizes.NumOutputs    = 2;    %输出个数，缺省为0
32      sizes.NumInputs     = 2;    %输入个数，缺省为0
33      sizes.DirFeedthrough = 0;   %是否存在直馈通道，1表示存在，0表示不存在
34      sizes.NumSampleTimes = 1;   %采样时间个数，至少是一个
35      sys = simsizes(sizes);      %返回size数据结构所包含的信息
36      x0  = [0 0 0 0];            %设置初始状态
37      str = [];                   %保留变量置空
38      ts  = [0 0];                %设置采样时间
39      simStateCompliance = 'UnknownSimState';
40
41  function sys=mdlDerivatives(t,x,u)
42      % 系统参数
43      h1 = 0.0308;
44      h2 = 0.0106;
45      h3 = 0.0095;
46      h4 = 0.2086;
47      h5 = 0.0631;
48      g = 9.8;
49
50      % 输入的驱动力矩
51      tau1 = u(1);
52      tau2 = u(2);
53
54      % 状态变量：位置和速度
55      q1 = x(1);
56      dq1 = x(2);
57      q2 = x(3);
58      dq2 = x(4);
59
60      % 计算惯性矩阵 M
61      m11 = h1 + h2 + 2*h3*cos(q2);
```

```
62      m12 = h2 + h3*cos(q2);
63      m21 = m12;
64      m22 = h2;
65
66      M = [m11, m12;
67           m21, m22];
68
69      % 计算科氏力和向心力矩阵 C
70      c11 = -h3*sin(q2)*dq2;
71      c12 = -h3*sin(q2)*(dq1 + dq2);
72      c21 = h3*sin(q2)*dq1;
73      c22 = 0;
74
75      C = [c11, c12;
76           c21, c22];
77
78      % 计算重力势能矩阵 G
79      g1 = h4*g*cos(q1) + h5*g*cos(q1+q2);
80      g2 = h5*g*cos(q1+q2);
81      G = [g1; g2];
82
83      % 驱动力矩向量
84      tau = [tau1; tau2];
85      dq  = [dq1; dq2];
86
87      % 计算加速度
88      % ddq = M^-1 * (tau - C*dq - G)
89      invM = inv(M);
90      ddq = (invM)*(tau - C*dq - G);
91
92      % 状态导数
93      dq1dot = ddq(1);
94      dq2dot = ddq(2);
95
96      % 返回状态导数
97      sys = [dq1; dq1dot; dq2; dq2dot];
98
99  function sys=mdlUpdate(t,x,u)
```

```
100     sys = [];                   %sys表示下一个离散状态，即x(k+1)
101
102 function sys=mdlOutputs(t,x,u)
103     sys = [x(1);x(3)];                  %sys表示输出，即y
104
105 function sys=mdlGetTimeOfNextVarHit(t,x,u)
106     sampleTime = 1;             %设置下一次采样时间是在1s以后
107     sys = t + sampleTime;       %sys表示下一个采样时间点
108
109 function sys=mdlTerminate(t,x,u)
110     sys = [];
```

2.2 串并联神经网络辨识模型搭建

2.2.1 二阶延时 (参数为默认值)

二阶延时的串并联神经网络的输入有 **6** 个，分别为关节 1、2 的力矩 tau_1、τ_2，系统实际输出上一时刻关节 1、2 的角度 $q_1(k-1)$、$q_2(k-1)$ 以及上上时刻关节 1、2 的角度 $q_1(k-2)$、$q_2(k-2)$

1、数据获取

首先需要利用我们已经搭建了的 S-FUnction 模块进行数据集的采集，即给定白噪声输入，将输出的值引入到 Matlab 的 Workspace 中，具体的 Simulink 的仿真图如下图所示

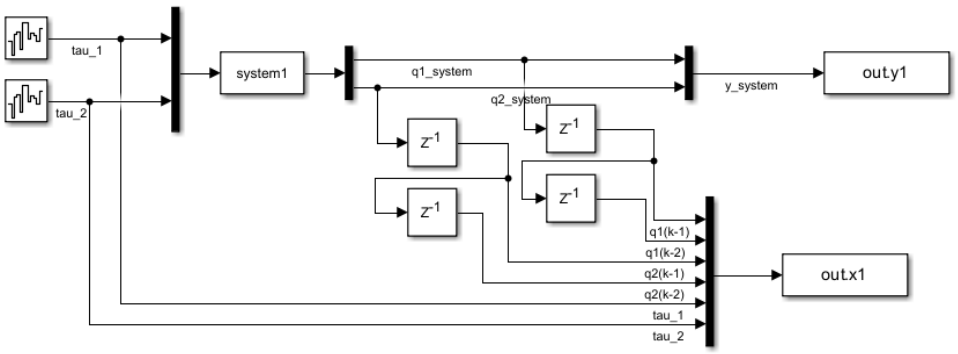


Figure 1: 二阶延时数据获取

在得到训练的数据之后,将其导入 **nftool** 模块中进行训练,这里采用了 **Levenberg-Marquardt** 法进行训练 (在实际测试过程中，发现其实几种方法的训练结构基本没有区别)，得到训练后的模型后，将其放入 Simulink 中，最终的仿真图如下图所示

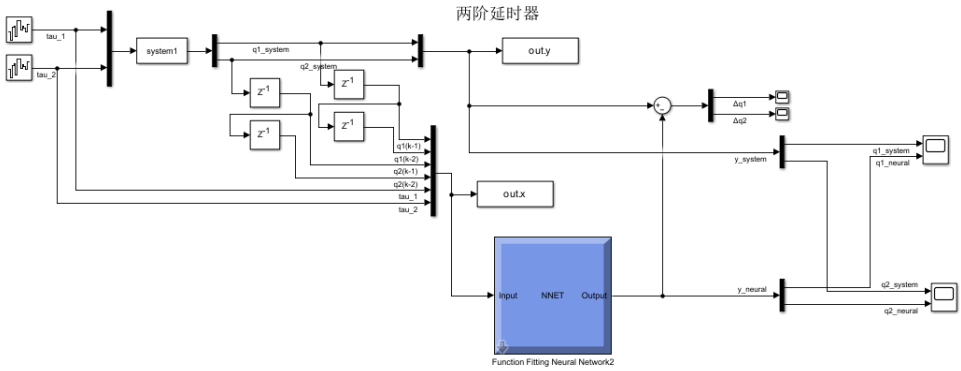
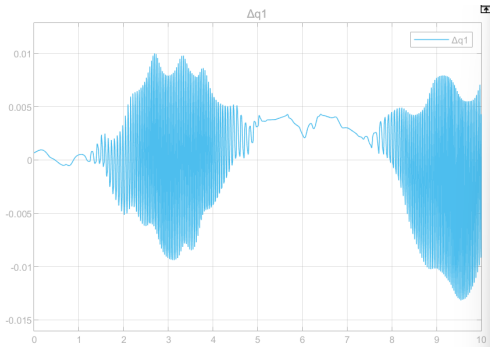
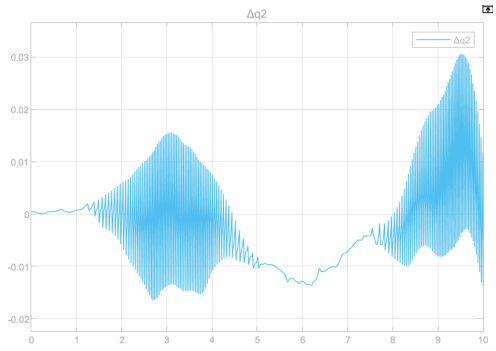


Figure 2: 二阶延时仿真图

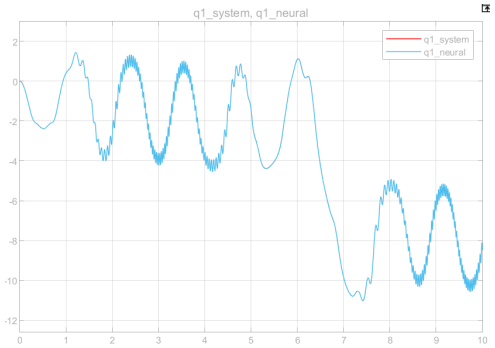
之后，将输入改成三角板与正弦波，得到实际输出与模型输出的结果及其差值分别如下图所示



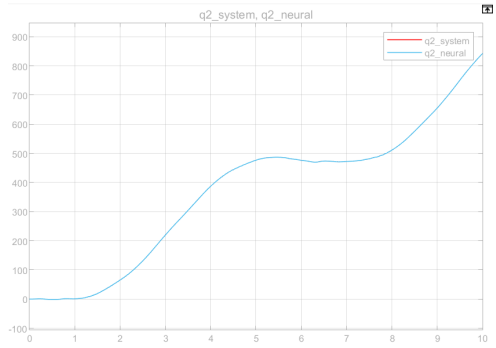
(a) δq_1



(b) δq_2



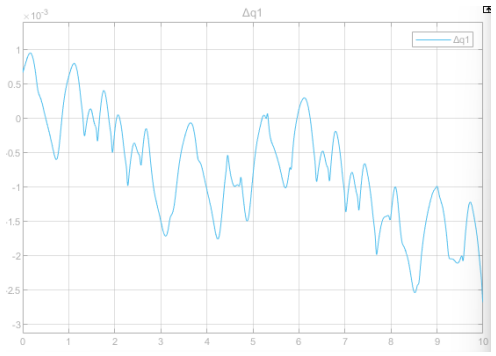
(c) q_1 的实际输出与系统输出对比



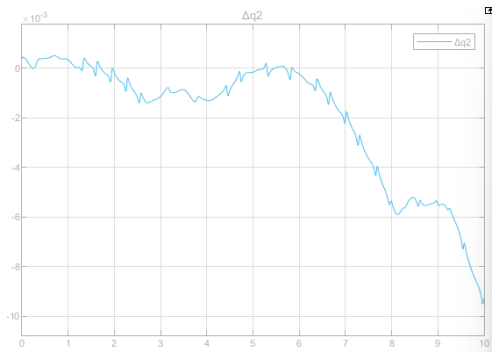
(d) q_2 的实际输出与系统输出对比

Figure 3: 正弦波输入

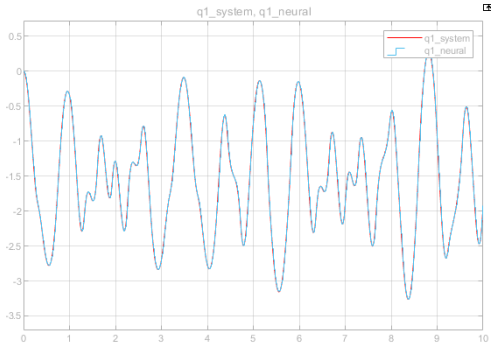
从图中可以看出，输出的两个角度值的差值均在 0.05 以内，且在输出对比中，可以看到神经网络的输出值能够很好的跟随上模型的实际输出，效果还不错。



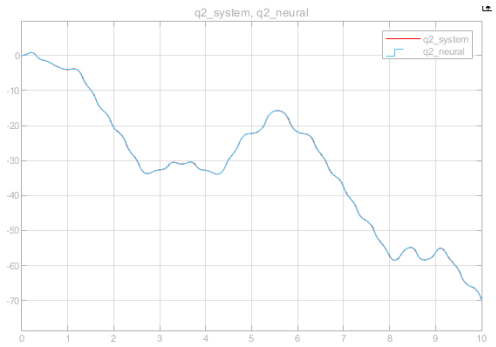
(a) δq_1



(b) δq_2



(c) q_1 的实际输出与系统输出对比



(d) q_2 的实际输出与系统输出对比

Figure 4: 三角波输入

从图中可以看出，虽然两个角度的误差曲线波动较大，但由于其是 10^{-3} 的数量级，因此可以忽略不记，从两者的实际曲线也可以看出基本上可以认为是重合的

2.2.2 三阶延时（参数为默认值）

从串-并联模型：（静态前向网络）的公式出发，考虑增加之前时刻的数据作为神经网络的输入，训练数据收集的流程与之前保持一致，构建的 Simulink 仿真图如下图所示

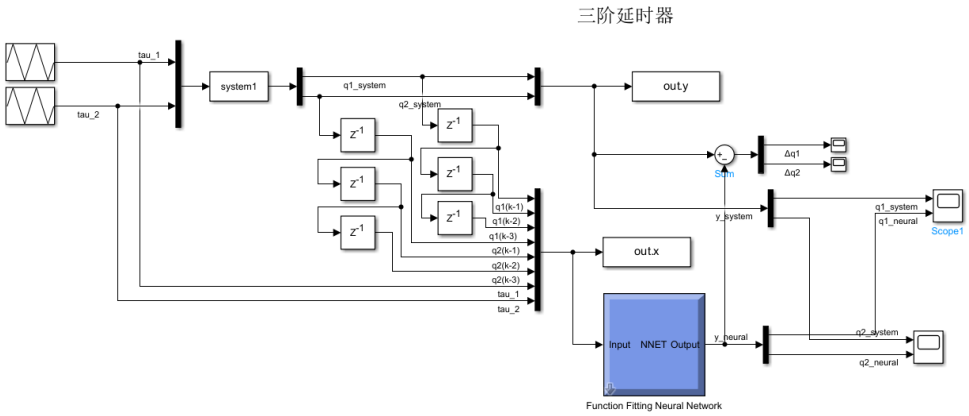


Figure 5: 三阶延时仿真图

三阶网络训练的到的模型的结果如下

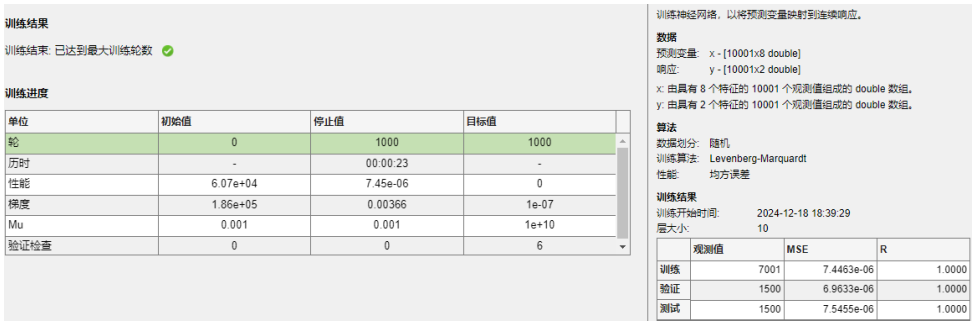
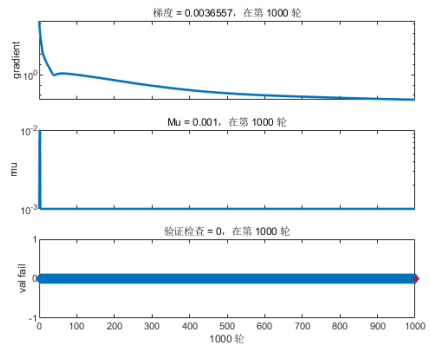
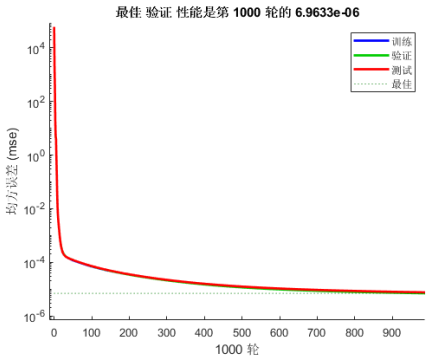


Figure 6: 三阶训练结果

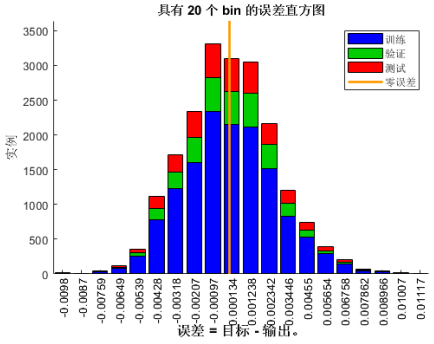
模型的各种指标如下图所示



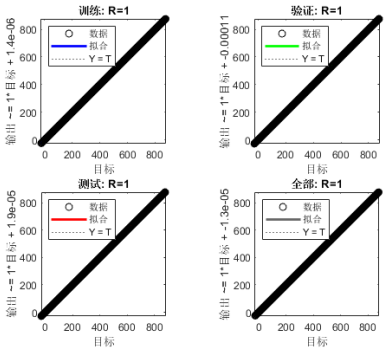
(a) 训练状态



(b) 性能



(c) 误差直方图

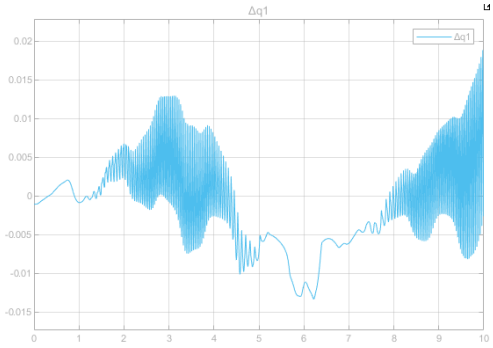


(d) 回归

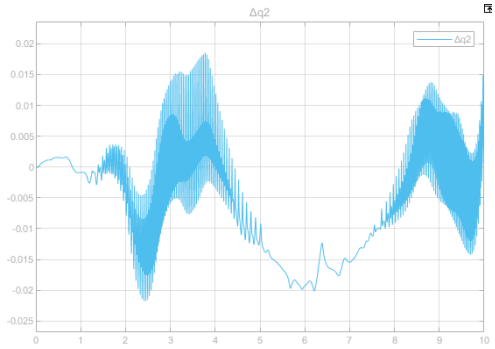
Figure 7: 模型各种指标

从上图可以看到梯度在开始时较大，随着训练的进行逐渐减小，即误差在减少。且训练集和测试集的误差都在快速下降，并且在较早的轮次就已经趋于平稳，且拟合较好，误差分布较集中。

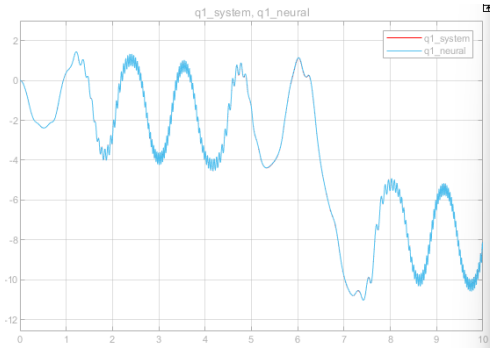
之后，给定三角输入与正弦输入，对比系统的实际输出与神经网络的输出，可以得到下图所示的结果。可以看到，无论是三角波输入还是正弦输入，其都能够很好的与实际的输出保持很小的偏差。



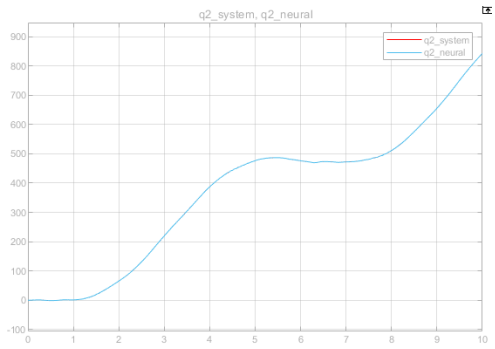
(a) δq_1



(b) δq_2

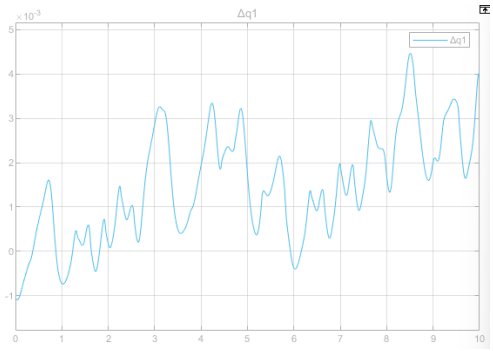


(c) q_1 的实际输出与系统输出对比

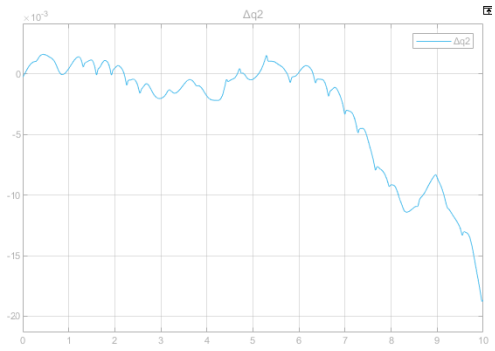


(d) q_2 的实际输出与系统输出对比

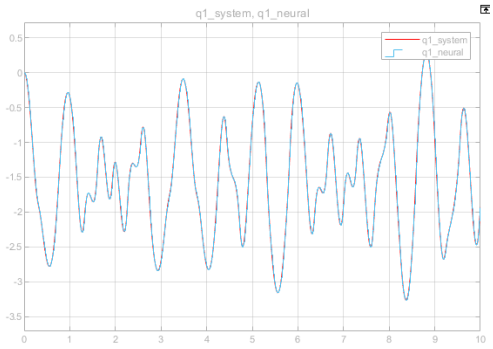
Figure 8: 正弦波输入



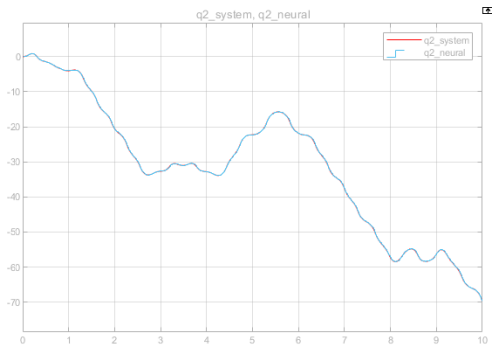
(a) δq_1



(b) δq_2



(c) q_1 的实际输出与系统输出对比



(d) q_2 的实际输出与系统输出对比

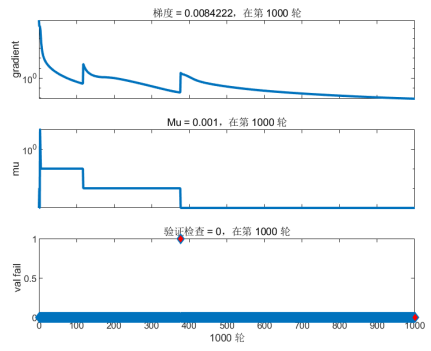
Figure 9: 三角波输入

2.2.3 三阶延时——增加不同种子值作为输入数据

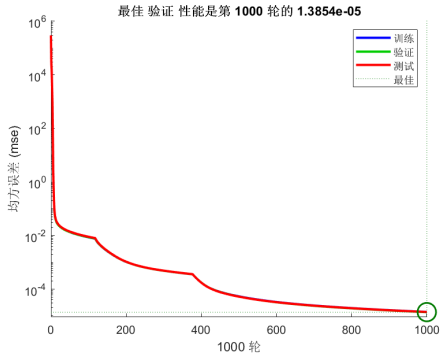
为了提高模型的泛化性，尝试提高数据量的值，并增加不同种子的值作为输入来获取数据，训练得到的模型的结果如下

Figure 10: 三阶训练结果

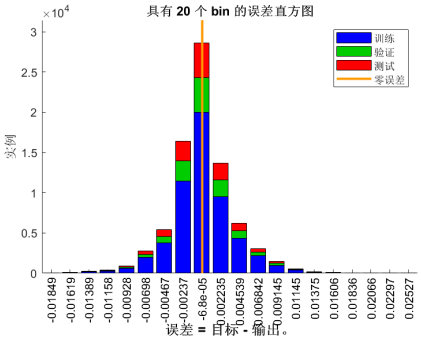
模型的各种指标如下图所示



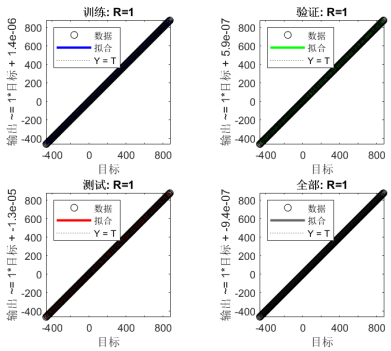
(a) 训练状态



(b) 性能



(c) 误差直方图

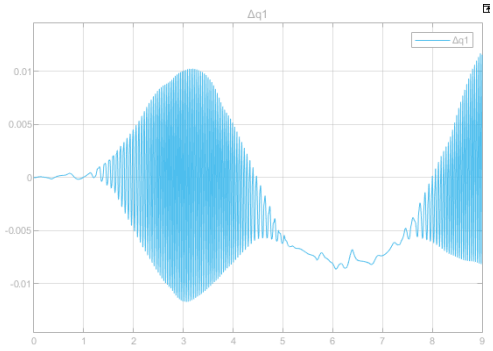


(d) 回归

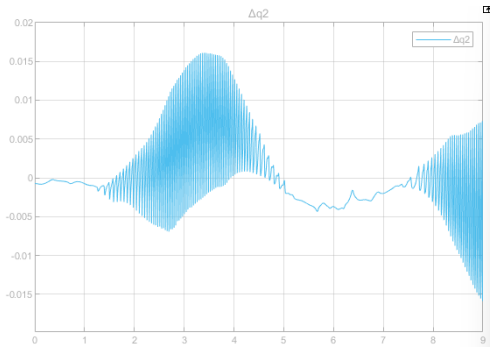
Figure 11: 模型各种指标

从上面可以看到，训练得到的模型的效果较好，练集和测试集的误差都在快速下降，并且在较早的轮次就已经趋于平稳，且拟合较好，误差分布较集中。

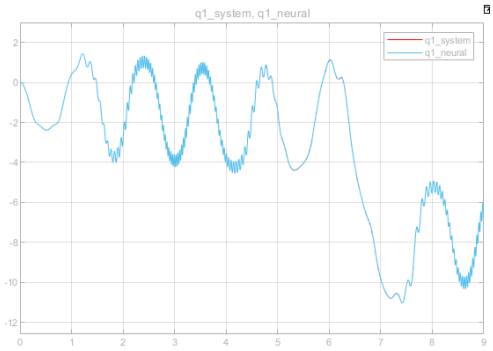
下面是在仿真中得到正弦波与三角波输入后的两个输出之间的比较



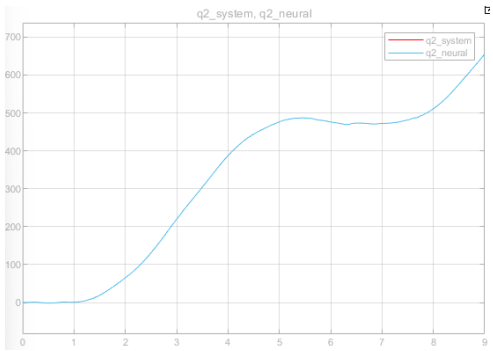
(a) δq_1



(b) δq_2

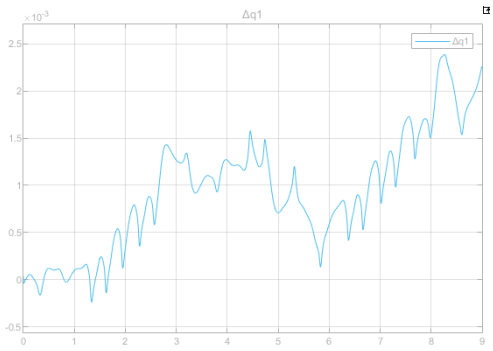


(c) q_1 的实际输出与系统输出对比

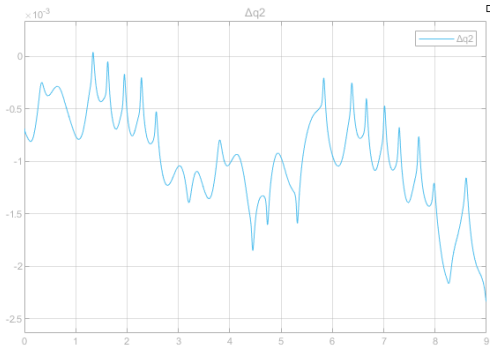


(d) q_2 的实际输出与系统输出对比

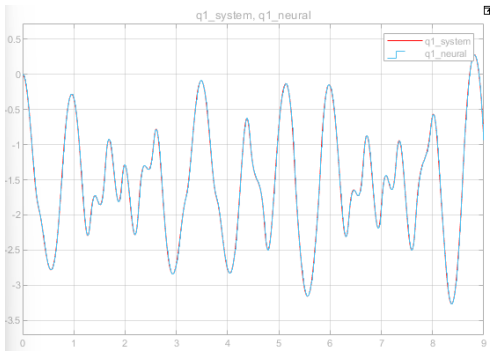
Figure 12: 正弦波输入



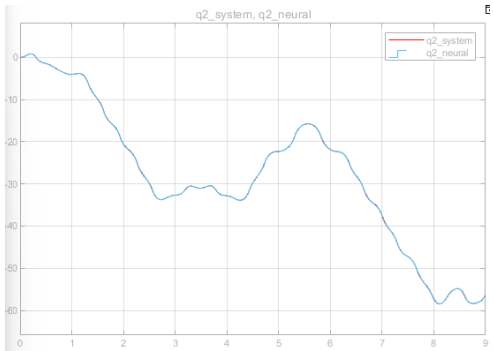
(a) δq_1



(b) δq_2



(c) q_1 的实际输出与系统输出对比



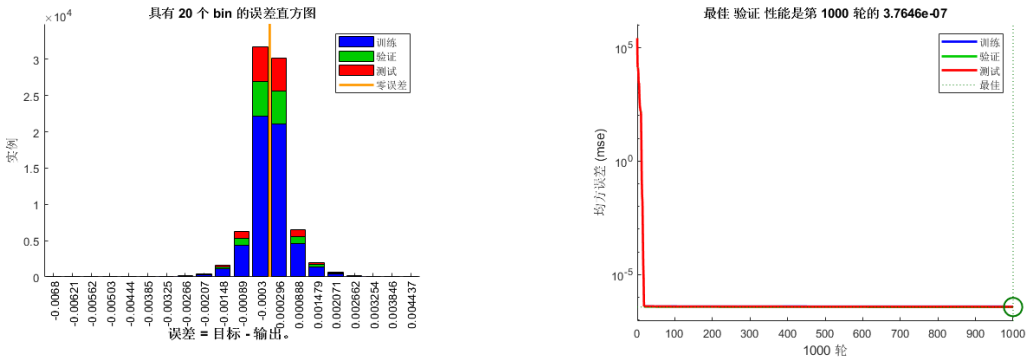
(d) q_2 的实际输出与系统输出对比

Figure 13: 三角波输入

从上图可以看到，无论是三角波还是正弦波，神经网络的输出波形与实际的输出波形都能够很好的保持一致，且误差极小。

2.2.4 三阶延时——增加数据且增加隐藏层个数

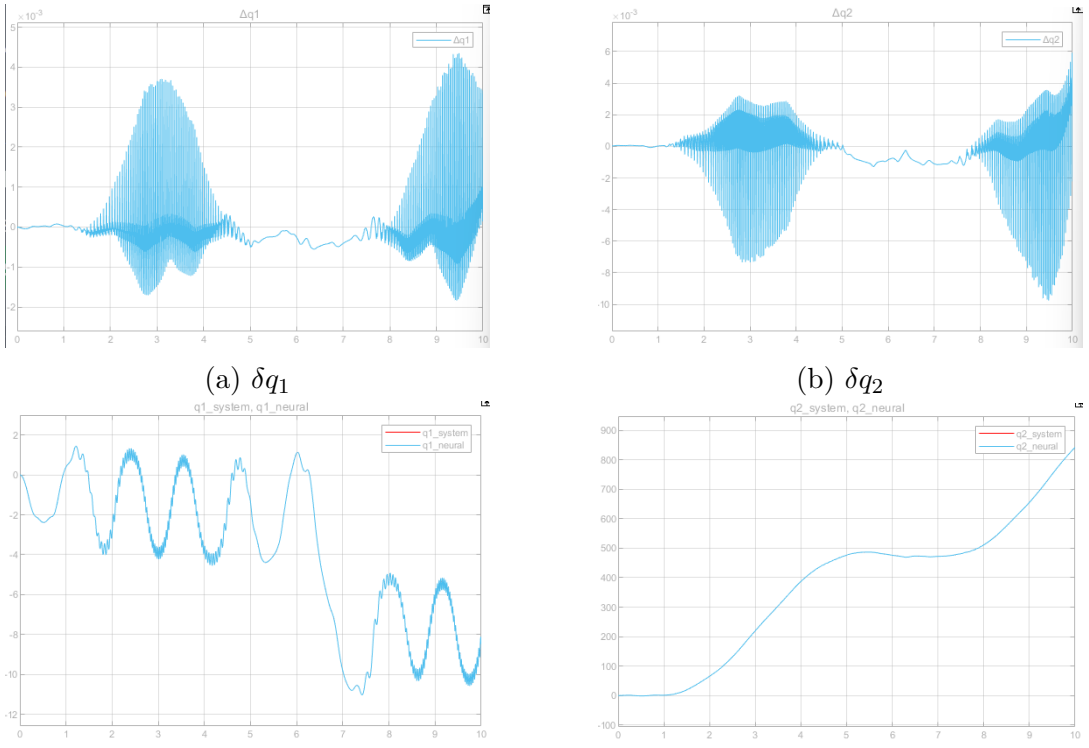
在上一步已经增加了不同种子的白噪声输入的数据后，提高训练模型的隐藏层个数到 20 个，训练得到的结果如下图所示



(a) 性能　　(b) 误差直方图

Figure 14: 模型各种指标

当为正弦波输入时，两个输出的结果如下图所示，可以看到其偏差值可忽略不记。

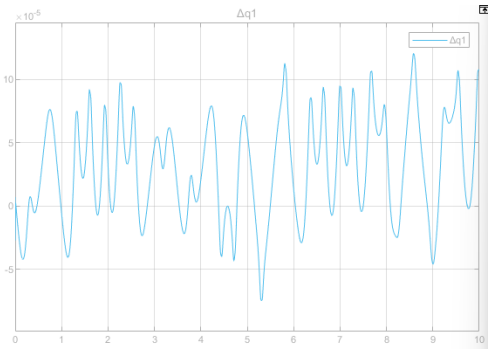


(a) δq_1　　(b) δq_2

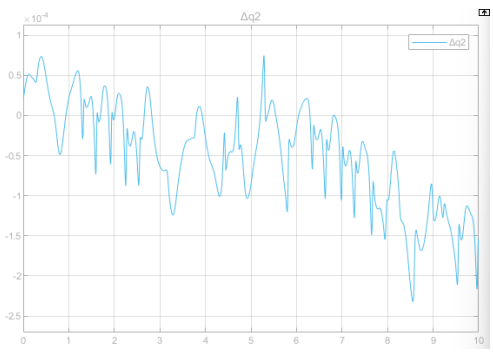
(c) q_1 的实际输出与系统输出对比　　(d) q_2 的实际输出与系统输出对比

Figure 15: 正弦波输入

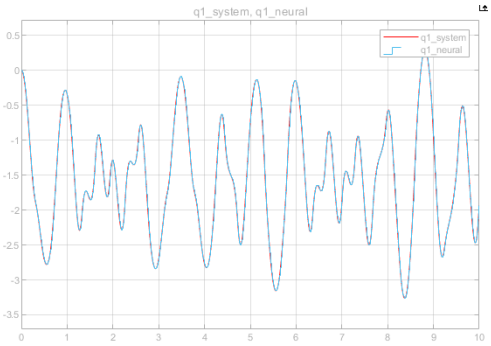
当为三角波输入时，两个输出的结果如下图所示，其偏差值也可以忽略不记。



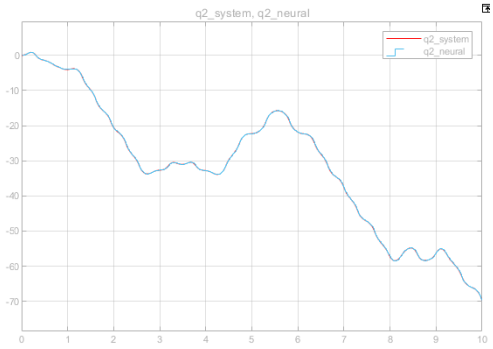
(a) δq_1



(b) δq_2



(c) q_1 的实际输出与系统输出对比



(d) q_2 的实际输出与系统输出对比

Figure 16: 三角波输入

2.2.5 对比分析

增加延时个数

在所有参其他变量均保持一致时，对比二阶延时与三阶延时的结果图，发现对于正弦输入与三角输入，发现其两个关节角的偏差均很小，偏差值也差不多，三阶有时最大值会大于二阶。这说明输入的过去时刻的值对于模型的训练的影响并没有太大

改变种子值提高数据量

当通过改变种子值来增加训练模型的数据后，无论是正弦波输入还是三角波输入，所有的偏差均有明显的减少，即神经网络训练得到的结果更加逼近于真实值

增加隐藏神经元个数

与 10 个神经员相比较，20 个神经员得到的模型与实际输出的值的偏差进一步降低，降低了一个数量级。效果更好

2.3 并联神经网络辨识模型搭建

下面构建了一个基于 Hopfield 神经网络的预测模型，其核心目标是通过给定的历史数据（过去几个时刻的系统输出）和外部输入（如力矩等）来预测系统在当前时刻的输出。为了达到这一目标，模型利用梯度下降法来优化神经网络的权重，从而最小化预测输出与实际输出之间的误差。

模型的输入由历史输出数据和外部输入信号组成，形成一个大小为 8x1 的向量：

$$\texttt{input} = \begin{bmatrix} \texttt{q1\_lllast} \\ \texttt{q2\_lllast} \\ \texttt{q1\_llast} \\ \texttt{q2\_llast} \\ \texttt{q1\_last} \\ \texttt{q2\_last} \\ \texttt{tau1} \\ \texttt{tau2} \end{bmatrix} \tag{14}$$

该向量包含了过去三时刻的输出数据以及当前外部输入信号，这些信息被用作神经网络的输入。

模型使用一个 8x2 的权重矩阵 $\mathbf{W}$，用于从输入向量生成预测输出。权重矩阵的初始值为零，并且通过梯度下降法进行优化。网络的目标是通过学习历史数据与系统实际输出之间的关系来更新权重，使得预测结果与实际结果尽可能接近。

为了优化神经网络的训练过程，学习率 η 被设定为一个初始值，并随着训练过程的进行逐步衰减。学习率的衰减采用指数衰减的方式，这有助于在训练后期减缓权重更新的速度，从而稳定网络的收敛：

$$\eta = \eta_{\text{initial}} \times (\text{decay\_rate})^{\frac{\text{count}}{\text{decay\_steps}}} \tag{15}$$

这里，η_{initial} 是初始学习率，decay_rate 是衰减率，decay_steps 控制了学习率衰减的速度。

在每次迭代中，模型会计算预测输出与目标输出之间的误差 error_last，并基于该误差通过梯度下降法更新权重。具体的梯度计算为：

$$\begin{cases} \text{grad} = -\text{input} \times \text{error\_last}^T \\ \mathbf{W} = \mathbf{W} - \eta \times \text{grad} \end{cases} \tag{16}$$

通过这种方式，权重矩阵 $\mathbf{W}$ 会逐步调整，使得神经网络的输出与目标系统输出之间的误差最小化。

为了避免数值不稳定，模型在每次迭代后都会对权重矩阵 $\mathbf{W}$ 进行归一化：

$$\begin{cases} \text{W\_norm} = \|\mathbf{W}\|_{\text{Frobenius}} \\ \text{if W\_norm} > 1e-6 \quad \text{then} \quad \mathbf{W} = \frac{\mathbf{W}}{\text{W\_norm}} \end{cases} \tag{17}$$

通过归一化，权重矩阵的范数被控制在合理的范围内，防止在训练过程中出现梯度爆炸等问题。

模型设置了两种提前终止的条件：

- **误差收敛条件**：当当前误差的 L2 范数小于设定阈值（如 1×10^{-6}）时，训练提前终止。
- **误差变化过小**：当当前误差与上次误差的变化小于设定的阈值时，训练提前停止。

这两种条件有助于避免不必要的训练步骤，并提高计算效率。

训练结束后，神经网络使用优化后的权重矩阵对当前输入进行预测：

$$\begin{cases} q = \mathbf{W}^T \times \text{input} \\ q1 = q(1); \quad q2 = q(2) \end{cases} \tag{18}$$

在 Simulink 中的仿真图如下图所示

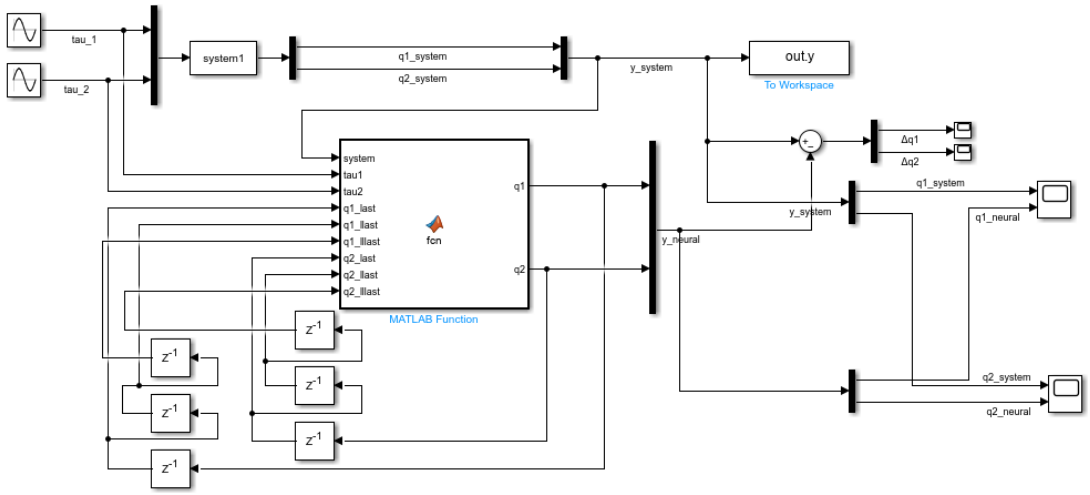
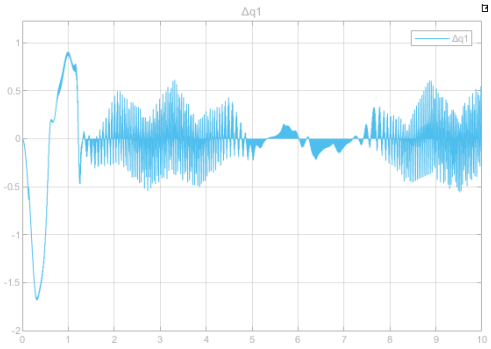
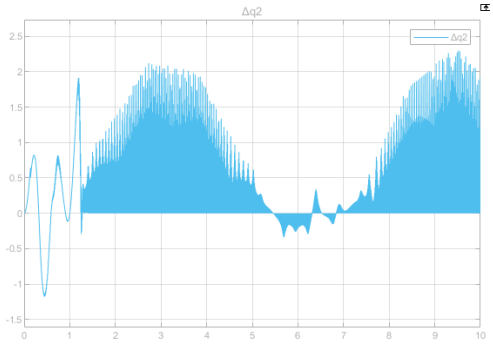


Figure 17: 并联模型仿真图

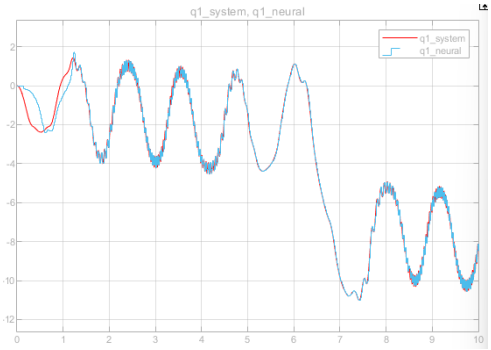
得到的结果如下图所示



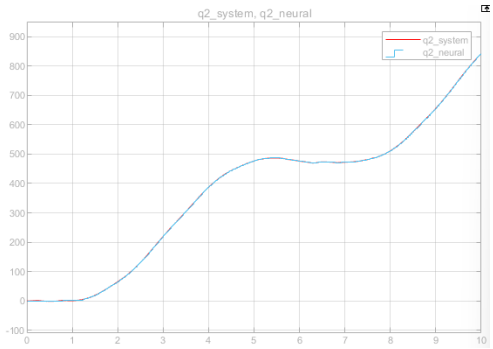
(a) δq_1



(b) δq_2



(c) q_1 的实际输出与系统输出对比



(d) q_2 的实际输出与系统输出对比

Figure 18: 正弦波输入

从上图可以看到在开始的时候误差较大，过一段时间后基本能够与实际输出保持一致，偏差较小。

3 文件列表

文件名	功能描述
report.pdf	实验报告
README.md	文件结构说明
code	代码文件夹
system1.m	S-Function 代码
second_order_delay _10level_model.slx	二阶延时仿真
third_order_delay_10level_model.slx	三阶延时仿真
multiples_white_noise_10level_model.slx	三阶延时、多种白噪声输入仿真
multiples_white_noise_20level_model.slx	三阶延时、多种白噪声输入、20 个隐藏神经元仿真
hopfiled_model.slx	Hopfiled 神经网络辨识模型仿真
out_x.mat	多种白噪声输入数据 x
out_y.mat	多种白噪声输入数据 y