Цель работы.

Нейросетевая аппроксимация СХ НЭ**.**

Основные теоретические положения.

Полностью определенные модели многих объектов и элементов систем управления не удается получить аналитическим способом. Тогда привлекаются процедуры идентификации, позволяющие оценивать параметры путем обработки данных экспериментов — активных (специально спланированных и реализованных) или пассивных (в режиме нормальной эксплуатации).

Нейросетевые модели НЭ рекомендуются в ситуации, когда тип нелинейности не известен, но имеются данные о ее входах и выходах. Искусственные нейронные сети (ИНС) представляют собой универсальные подстраиваемые аппроксиматоры безынерционных многомерных НЭ с однозначной СХ.

На рис. 3.1 изображена модель нейрона. Выходная переменная y является нелинейной функцией взвешенной суммы входных переменных

$$y = F(b + \sum_{i=1}^{n} w_i x_i),$$

, где: x_i — сигнал на входе нейрона, w_i — вес i-го входа, b — смещение; F(s) — функция активации. Параметры нейрона — смещение b и весовые коэффициенты wi можно настраивать, добиваясь требуемой зависимости выхода от входов.

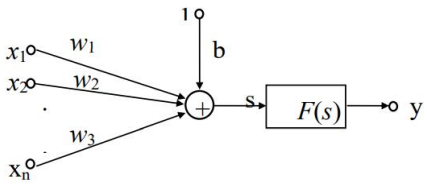

Рис. 3.1. Модель нейрона

Широкие аппроксимирующие возможности достигаются, если нейроны образуют сети определенной архитектуры. На рис. 3.2 изображена однонаправленная двухслойная нейронная сеть с тремя входами и двумя

выходами. Весовые коэффициенты и смещения первого (скрытого нелинейного) слоя сети упорядочены в матрицы **W1**, **b1**, а второго линейного — в матрицы **W2, b2**.

Разработан ряд алгоритмов обучения многослойных сетей — настройки весов и смещений из условия минимизации суммы квадратов отклонений выхода сети от выхода моделируемого объекта [8].

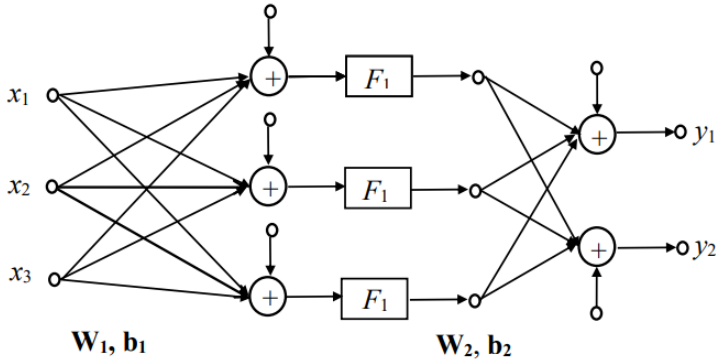

Рис. 3.2. Двухслойная нейронная сеть

Нейросетевая аппроксимация СХ многомерного НЭ на примере ГДХ центробежного компрессора природного газа.

Зависимость степени сжатия газа ε = p_вых/p_вх от объемного расхода Q и относительной частоты вращения компрессора n

$$\varepsilon = F\_1(Q, n) \qquad (3.1)$$

задается так называемыми газодинамическими характеристиками (ГДХ). Функция двух аргументов (3.1) задает СХ НЭ с двумя входами и одним выходом. Построенные по экспериментальным данным графики функции (3.1) приводятся в справочниках заводов-изготовителей (см., например, [9]). В качестве параметра семейства кривых обычно выступает относительная частота вращения ротора компрессора n.

Графики ГДХ наглядно отражают взаимосвязь основных переменных процесса компримирования газа, однако для разработки компьютерных (имитационных) моделей их следует привести к иной форме.

Существует множество способов аппроксимации нелинейных характеристик.

Методика построения нейросетевой модели складывается из следующих этапов:

1 — подготовка обучающих данных;

2 — выбор типа и архитектуры ИНС;

3 — обучение сети;

4 — анализ сети.

1. Подготовка обучающих данных на входе (наборы значений расхода Q и n) и выходе (набор значений степени сжатия ε) требует перевода ГДХ в табличную форму задания, т. е. оцифровки графиков ГДХ.

2. Выберем инструмент обучения ИНС — python(pytorch). Вводим данные для обучения сети.

Частота вращения n =[0.7 0.7 0.7 0.7 0.7 0.7 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.85 0.85 0.85 0.85 0.85 0.85 0.85 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.95 0.95 0.95 0.95 0.95 0.95 0.95 0.95 1 1 1 1 1 1 1 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.05 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1];

Объемный расход Q = [245 280 340 400 440 465 265 320 360 400 440 460 495 280 320 360 420 480 520 530 300 360 400 440 480 520 560 315 360 400 440 480 520 560 600 330 380 420 460 500 540 580 620 630 350 420 450 500 540 580 620 665 370 420 480 525 550 580 620 660 695 385 420 460 490 510 560 600 640 680 730];

Степень сжатия Epsilon = [1.145 1.140 1.13 1.11 1.09 1.08 1.166 1.16 1.15 1.14 1.12 1.11 1.08 1.19 1.185 1.18 1.16 1.13 1.105 1.095 1.22 1.21 1.20 1.185 1.17 1.144 1.11 1.246 1.24 1.232 1.22 1.205 1.183 1.155 1.12 1.276 1.27 1.26 1.25 1.23 1.21 1.18 1.145 1.135 1.31 1.3 1.29 1.27 1.25 1.23 1.195 1.15 1.345 1.34 1.32 1.30 1.29 1.27 1.245 1.21 1.165 1.382 1.38 1.37 1.36 1.35 1.33 1.305 1.275 1.24 1.18];

3. Обучение.

4. Анализ нейросетевой модели компрессора.

Экспериментальные результаты.

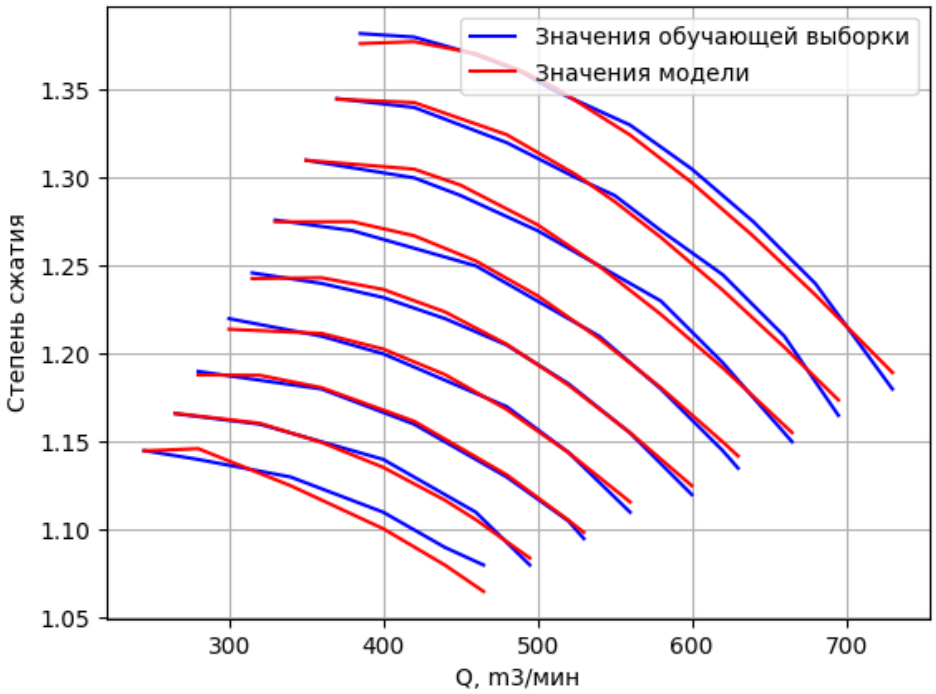

Рис 3.3 ГДХ компрессора (лучший вариант)

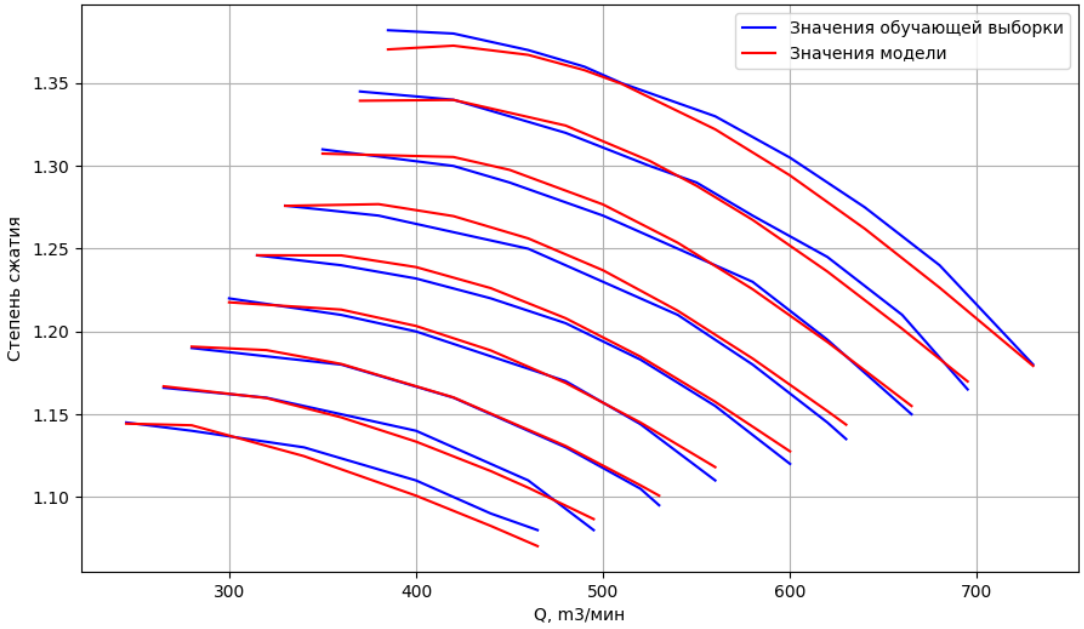

Рис. 3.4 ГДХ компрессора (hidden_size = 100, epoch_iterations = 3000)

Анализ работы.

1 этап. Подготовка обучающих данных.

```
#Вектор входных значений
def __create_training_data__(self):
        # Массив входных значений
        self.x = torch.tensor([n,Q])
        self.x = torch.transpose(self.x, 0, 1)


        # Обучающий вектор выходных значений
        self.y = torch.tensor(Epsilon)
        self.y = torch.torch.unsqueeze(self.y, 1)
```

В качестве входных данных формирую тензор размерности 2, состоящий из массивов n и Q. В качестве выходных данных беру тензор размерности 1, состоящий из массива Epsilon.

2 этап. Выбор типа и архитектуры ИНС.

Для нашей работы мы выбрали вид ИНС Feedforward, что означает, что нейронная сеть прямого распространения. Также наша нейронная сеть многослойная: состоит из двух линейных слоев, и в скрытом слое содержит 100 нейронов. Другими словами, выбрали модель двуслойного перцептрона. В качестве функции активации сигмоида – гладкая и непрерывная, что пригодится нам при использовании back propogation.

```
class Feedforward(torch.nn.Module):
        '''
         input size - размер входных данных
        output_size - размер выходных данных
         hidden size - количество нейроннов в первом слое
         '''
        def __init__(self, input_size, output_size, hidden_size):
                super(Feedforward, self).__init__()
                self.input_size = input_size
                self.hidden_size = hidden_size
                self.output_size = output_size

        # Первый слой нейронной сети
        # Linear задает следующее преобразование:
        # y = x*A^T + b
```

```
        # x - входной вектор, y - выходной вектор
        # A^T - транспонированная матрица весов слоя
        # b - сдвиг слоя


                self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
                # Второй слой нейронной сети
                self.fc2 = torch.nn.Linear(self.hidden_size, self.output_size)
                # Функция активации нейронной сети, выбрана сигмоида:
                # f(x) = 1/(1+x^-1)
                self.sigmoid = torch.nn.Sigmoid()
'''
  Функция расчета значений нейронной сети по входному вектору
  ко входному вектору последовательно применяются преобразования,
  каждого слоя нейронной сети
  '''
        def forward(self, x):
                hidden = self.fc1(x)
                output_sig = self.sigmoid(hidden)
                output = self.fc2(output_sig)
                return output
```

3 этап. Обучение сети.

```
def __train_model__(self, epoch_iterations=0):
        # Нужно вызвать перед тренировкой модели
        self.model.train()

        # Количество итераций обучения
        for epoch in range(epoch_iterations):
                self.optimizer.zero_grad() #обнуляю накопленные градиенты
        # Считаем значений на текущих коэффициентах модели
                self.y_model_output = self.model(self.x)
                # Считаем значение ошибки относительно значений обучающей выборки
                loss = self.criterion(self.y_model_output.squeeze(), self.y.squeeze())
                print('Epoch {}: train loss: {}'.format(epoch, loss.item()))


                # Вызываем итерацию настройки коэффицентов
                 loss.backward() # считаю градиенты
                self.optimizer.step() # делаю обновление

         self.y_model_output = self.y_model_output.detach().numpy()
```

На этом этапе я обучаю нашу модель. Обучение происходит в два этапа: прямое распространение ошибки и обратное распространение ошибки. Во время прямого распространения ошибки делается предсказание ответа. При обратном распространении ошибка между фактическим ответом и предсказанным минимизируется. В качестве критерия обучения выбираем среднеквадратичную ошибку MSE, а в качестве алгоритма настройки выбираем алгоритм обратного распространения ошибки back propogation. Алгоритм использует так называемый стохастический градиентный спуск, «продвигаясь»

в многомерном пространстве весов в направлении антиградиента с целью достичь минимума функции ошибки.

На данном этапе нужно было подобрать оптимальное число эпох epoch_iterations. Я воспользовалась методом локтя и получила значение примерно равное 250. Однако на практике, при таком количестве эпох предсказанные значения выродились в вертикальную прямую. После этого я методом проб и ошибок подобрала значение 3000.

4 этап. Анализ сети.

На 4 этапе я рисую значения обучающей выборки (синим) и полученные значения (красным) по соответствующим значениям n.

```
def __draw_the_network_output__(self):
        plt.figure("График сравнения модели")
        plt.grid()
        plt.ylabel('Степень сжатия')
        plt.xlabel('Q, m3/мин')

        # Нарисовать значения обучающей выборки(синим)
        plt.plot(self.x[:6, 1], self.y[:6], color="blue", label='Значения обучающей выборки')
        plt.plot(self.x[6:13, 1], self.y[6:13], color="blue")
        plt.plot(self.x[13:20, 1], self.y[13:20], color="blue")
        plt.plot(self.x[20:27, 1], self.y[20:27], color="blue")
        plt.plot(self.x[27:35, 1], self.y[27:35], color="blue")
        plt.plot(self.x[35:44, 1], self.y[35:44], color="blue")
        plt.plot(self.x[44:52, 1], self.y[44:52], color="blue")
        plt.plot(self.x[52:61, 1], self.y[52:61], color="blue")
        plt.plot(self.x[61:, 1], self.y[61:], color="blue")
        # Нарисовать значения модели
         plt.plot(self.x[:6, 1], self.y_model_output[:6], color="red", label='Значения модели')
        plt.plot(self.x[6:13, 1], self.y_model_output[6:13], color="red")
        plt.plot(self.x[13:20, 1], self.y_model_output[13:20], color="red")
        plt.plot(self.x[20:27, 1], self.y_model_output[20:27], color="red")
        plt.plot(self.x[27:35, 1], self.y_model_output[27:35], color="red")
        plt.plot(self.x[35:44, 1], self.y_model_output[35:44], color="red")
        plt.plot(self.x[44:52, 1], self.y_model_output[44:52], color="red")
        plt.plot(self.x[52:61, 1], self.y_model_output[52:61], color="red")
        plt.plot(self.x[61:, 1], self.y_model_output[61:], color="red")


        plt.legend()
        plt.show()
```

По итогу видим, что наша ИНС достаточно хорошо аппроксимирует СХ НЭ.

Выводы.

По итогу работы мы изучили feedforward нейронную сеть, которая помогла нам аппроксимировать СХ НЭ.