CHALMERS
UNIVERSITY OF TECHNOLOGY

Performance prediction method of vehicle integrated thermal system based on Neural Network

Master's thesis in Electric Power Engineering

ZICHANG HUANG

DEPARTMENT OF ELECTRIC POWER ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Performance prediction method of vehicle integrated thermal system based on Neural Network

ZICHANG HUANG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electric Power Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Performance prediction method of vehicle integrated thermal system based on Neural Network
ZICHANG HUANG

© ZICHANG HUANG, 2022.

Supervisor: Akshay; Christoffer Strömberg, Engineers at CEVT
Examiner: Torbjörn Thiringer, Professor, Electrical Engineering Department

Master's Thesis 2022
Department of Electric Power Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Department of Electric Power Engineering
Gothenburg, Sweden 2022

Performance prediction method of vehicle integrated thermal system based on Neural Network
ZICHANG HUANG
Department of Electric Power Engineering
Chalmers University of Technology

Abstract

An important index of electric vehicles is the vehicle endurance, and the optimal design of vehicle integrated thermal system(TEM) will greatly improve the energy efficiency of vehicles. However, using experimental methods to design and optimize TEM requires a lot of manpower and time investment, so most of the development and testing are simulated by physics based models. Physical models also have some problems, such as cumbersome simulation steps, complex model structure and time-consuming, which are difficult to flexibly meet the development requirements. Therefore, in this study, the neural network model is used to predict the TEM performance, which reduces the complexity of the development process and improves the development efficiency.

Specifically, the neural network model is trained by using the prototype vehicle experimental data accumulated in the early stage of development, and a part of the experimental data is divided into independent test sets to test the model, so as to determine the prediction accuracy of the model. Through comparison, the GRU model with self attention mechanism is used as the final neural network prediction model, and shows high prediction accuracy for the performance parameters of TEM components during testing.
Keywords: Neural network, vehicle integrated thermal system, time series forecasting, recurrent neural network, long-short term memory model, gated recurrent unit model, self attention mechanism.

Acknowledgements

This may be my last work as a student. I hope this thesis can draw a complete end to my student life. The work of this thesis may not be of great significance, but for me personally, it may become an interesting memory many years later.

When I explored the thesis project in the company, I thanked Christoffer and Akshay for their suggestions and help, and also thanked Torbjörn Thiringer, for reviewing and revising my thesis. In addition, I also thank my friend Songyue Wei for providing me with guidance and explanation on neural network. I would also like to thank CEVT for providing this thesis project opportunity. The work of this project has changed my employment direction to a certain extent.

Finally, I want to summarize it in a poem: 别离难,不似相逢好

Zichang Huang, Gothenburg, June 2022

Nomenclature

ANN	Artificial Neural Network
bi-LSTM	Bidirectional Long-Short Term Memory
CAE	Computer-aided Engineering
CNN	Convolutional Neural Network
FCN	Fully-Connected Neural Network
FTP-75	The EPA Federal Test Procedure
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
MHA	Multi-Head Attention
NEDC	New European Driving Cycle
Relu	Rectified Linear Units
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SDPA	Scaled Dot-Product Attention
Seq2Seq	Sequence to Sequence
SGD	Stochastic Gradient Descent
Tanh	Hyperbolic Tangent
TEM	Thermal System Model
WLTC	World-wide harmonized Light duty driving Test Cycle

Contents

1 Introduction **1**
1.1 Project background 1
1.2 Research object 2
1.3 Experimental data and research platform 3
1.3.1 Input and output for neural network training 3
1.3.2 Experimental data source 4
1.3.3 Dataset classification and training platform 5

2 Principle of artificial neural network **7**
2.1 Principle and structure of artificial neuron 7
2.2 Fully-Connected Neural Network(FCN) 8
2.2.1 Forward propagation derivation 8
2.2.2 Backward propagation derivation 10
2.3 Recurrent Neural Network(RNN) 12
2.3.1 Original Recurrent Neural Network 12
2.3.2 Long-Short Term Memory model 13
2.3.3 Gated Recurrent Unit model 16
2.4 Transformer 19
2.4.1 Transformer introduction 19
2.4.2 Principle of attention mechanism 20
2.4.3 Principle of self-attention mechanism 22
2.4.3.1 Encoder self-attention module: 23
2.4.3.2 Decoder self-attention module 24
2.4.3.3 Encoding-decoding attention module 24

3 Neural network training process and strategy **27**
3.1 Determine the hyper-parameter of neural network 27
3.1.1 Input data normalization 27
3.1.2 Method for determining the architecture of neural network . . 29
3.1.2.1 Determine the number of hidden layers 29
3.1.2.2 Determine the number of neurons in hidden layer . . 30
3.1.2.3 Selection of activation function 32
3.1.3 Determine the learning rate of neural network 35
3.1.4 Drop out and gradient clipping 37
3.1.5 Training epoch number and early stop mechanism 38

3.2 Problems encountered in model training and Countermeasures 40
3.2.1 The influence of batch size on the prediction accuracy of the model 40
3.2.1.1 Reasons for adopting large batch size 40
3.2.1.2 Problems encountered in using large batch size . . . 41
3.2.1.3 Solution of large batch size problems 42
3.2.2 The problem of output weight assignment under multiple output task 43
3.3 Determine the type of neural network used for the project 45

4 Neural network prediction results and comparison 49
4.1 Prediction of power consumption of vehicle thermal system 50
4.2 Prediction of inlet pressure of water cooling condenser 51
4.3 Prediction of average temperature of evaporator 52
4.4 Prediction of average temperature of condenser 53
4.5 Prediction of compressor inlet temperature 54
4.6 Prediction of compressor inlet pressure 55
4.7 Prediction of radiator inlet temperature 56
4.8 Prediction of chiller inlet temperature 57

5 Discussion 59
5.1 Future work 59
5.2 Ethical and sustainability impacts of the project 59

6 Conclusion 61

1 Introduction

1.1 Project background

With the increasing popularity of electric vehicles, automobile manufacturers are comprehensively promoting the electrification, energy conservation and intelligence of automobile architecture, in order to obtain advantages and lead innovation in the fierce competition. For electric vehicles, the endurance anxiety caused by charging time and other reasons makes the endurance mileage a very important performance index. Therefore, automobile manufacturers invest a lot of resources to optimize energy structure and design complex energy recovery system to improve automobile energy efficiency. Among them, the design of an efficient automobile thermal system is the key part to improve energy efficiency [1].

Traditionally the vehicle homologation is done for one ambient temperature only, so the published values of driving range and energy consumption corresponding to that temperature. However, when a customer is driving his/her car on the road, the energy consumption and range may be significantly different due to many factors, among others the energy consumption from the thermal management of vehicle components, and from the cabin climate system. Therefore, it is necessary to test the vehicle energy consumption and endurance performance at different temperatures. However, due to the large experimental time and labor cost, most of the tests are carried out through simulation software. The traditional simulation platform usually has the disadvantages of large difference from the experimental results and cumbersome modeling steps, so a new vehicle performance prediction method is needed [2, 3].

In the early stage of automobile development, a large number of experimental data of systems, subsystems or components have been accumulated. The existing vehicle transient driving unit test data are used to train the neural network, and the trained neural network model is used as an alternative method of performance prediction, so as to reduce the dependence on experiments and solve the potential problems of traditional simulation models.

1.2 Research object

The research object of this project is the integrated thermal system of a prototype vehicle in the development stage. The thermal system is designed to improve comprehensive energy efficiency, so two operation modes are designed for different working conditions.

Figure 1.1 shows the structure of the joint simulation platform. The vehicle model is responsible for simulating the transient conditions of the vehicle environment, such as road conditions, ambient temperature, cabin temperature, etc. The controller model is responsible for simulating the control strategy and control signal made by the controller according to the environment of the vehicle. Then, the thermal system model(TEM) simulates the real performance of various thermal system components under this condition according to the received controller signal. At present, the physical method is used to simulate the performance of components.

The goal of this project is to apply neural network to predict the actual performance of the whole thermal system and various components, and shorten the cpu speed, and integrate it into the development platform for joint simulation with the simulation models of other components.

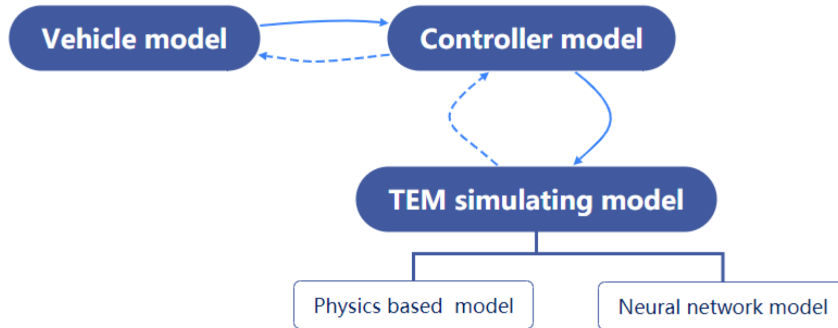


Figure 1.1: Structure of the joint simulation platform

1.3 Experimental data and research platform

1.3.1 Input and output for neural network training

Neural network learns and predicts thermal system performance from the existing experimental data, and its learning method is to determine the causal law between input and output, rather than calculating the output from the input according to the laws of physics like the physics-based model. Therefore, the control signal sent by the controller model can be used as the input parameters to find out the causal relationship between the control signal and the actual performance of components through training the neural network model, so as to omit the complex physical changes from control to actual operation and directly predict the operation results. The input interface for neural network training are shown in Table 1.1.

Table 1.1: Neural network training input interface

Input parameter			
#1	Vehicle speed	#8	Request speed of compressor
#2	Ambient temperature	#9	Request speed of ED pump
#3	Cabin temperature	#10	Request speed of battery pump
#4	Mode	#11	Average battery temperature
#5	Request speed of cold blow fan	#12	Request speed of evaporator expansion valve
#6	Request speed of warm blow fan	#13	Request speed of chiller expansion valve
#7	Request speed of cooling fan		

The output interface for neural network training are shown in Table 1.2.

Table 1.2: Neural network training output interface

Output parameter			
#1	Thermal system power	#5	Inlet pressure of water cooling condenser
#2	Inlet temperature of compressor	#6	Inlet pressure of compressor
#3	Average temperature of evaporator	#7	Inlet temperature of radiator
#4	Average temperature of condenser	#8	Inlet temperature of chiller

1.3.2 Experimental data source

The training data used for neural network is the data collected in the experiment in the early stage of development. The experimental data is collected by the corresponding position sensor when the equal proportion engineering sample vehicle is tested in the professional test laboratory. The sample vehicle is based on World-wide harmonized Light duty driving Test Cycle (WLTC) standard, and is tested for dozens of times and collected data under different environmental conditions (soaked/steady, different climate, etc.). Figure 1.2 and 1.3 show experimental data samples for the input and output of neural network training. (Reminder: since the experimental data is required to be confidential, the detailed coordinates and units are not provided.)

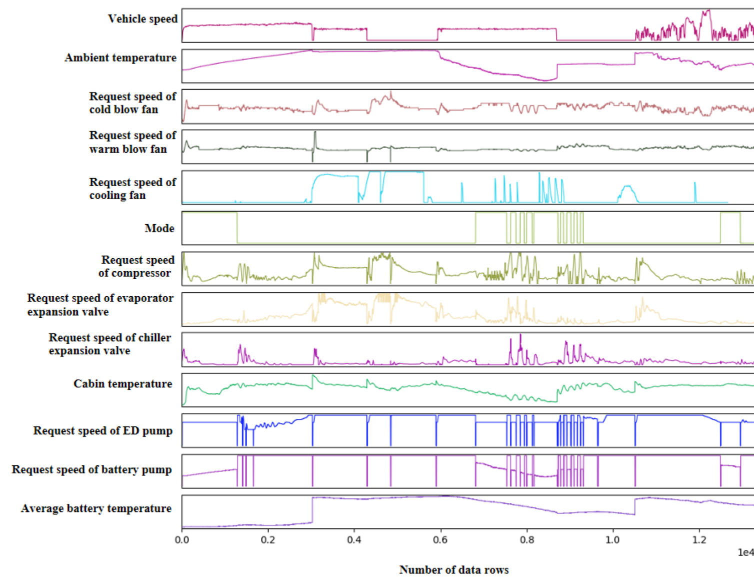


Figure 1.2: Input data set for neural network training

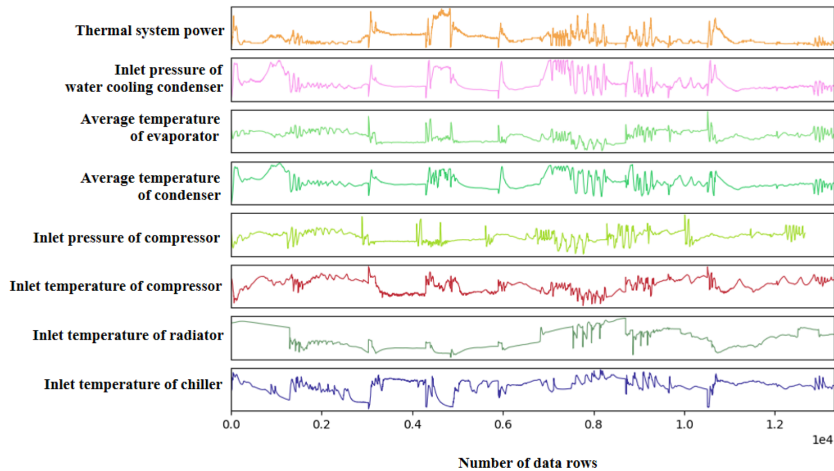


Figure 1.3: Output data set for neural network training

1.3.3 Dataset classification and training platform

Neural network needs a lot of data for training, and in the training process, according to different needs, the data set is divided into three categories: training set, validation set and test set [4].

Firstly, the data set is divided into training set and test set. Because the model construction process also needs to test the model, test the configuration of the model, as well as the training degree, over fitting or under fitting, the training data will be divided into two parts. One part is the training set for training and the other part is the Validation set for verification. Validation sets can be reused and are mainly used to assist in building models.

The training set is used to train the neural network model, and then the validation set is used to verify the effectiveness of the model, and the model with the best effect is selected until we get a satisfactory model. Finally, after the model "passes" the validation set, we use the test set to test the final effect of the model and evaluate the accuracy and error of the model. The test set is only used for model verification. It is absolutely forbidden to adjust the network parameter configuration and select the trained model according to the results of the test set, otherwise the model will be over fitted on the test set. Generally speaking, the final accuracy of the training set is greater than the validation set, and the validation set is greater than the test set.

When dividing these three types of data sets in proportion, the usual empirical proportion is 8-1-1, 7-2-1, 6-2-2 and 5-3-2. In the practice of the project, the division ratio is 7-1-2, that is, 70% of the data is used for training, 10% for validation and 20% for testing.

In terms of neural network training framework, the current mainstream machine learning framework includes Tensorflow platform and Pytorch platform based on Python language. Most machine learning users carry out relevant development based on these two platforms, and MATLAB also provides neural network toolbox. In this project, Tensorflow 2.0 is finally adopted as the training platform of neural network.

In terms of hardware, neural network training has very high requirements for computing performance, and because its parallel computing characteristics accord with the structural characteristics of graphics card, the training of neural network model mainly depends on graphics card. Therefore, the graphics card with strong performance is the key to determine the training speed of neural network. The series of graphics cards launched by NVIDIA company are equipped with CUDA core, which can accelerate the scientific calculation related to machine learning. Therefore, the company's products are widely used in the industry. In the model training of this project, an NVIDIA V100 professional AI card is mainly used for calculation, and AMD ryzen 7 4800 is used as the solution CPU.

2

Principle of artificial neural network

2.1 Principle and structure of artificial neuron

Artificial neural network(ANN) is a computational model and system simulating the human brain. In the brain, each neuron is connected with other neurons through dendrites and axon terminals. When the amount of stimulation received by a neuron exceeds a certain threshold, the neuron will trigger and transmit signals to other secondary neurons. At the numerical level, artificial neural network models the nervous system of the biological brain through a group of nodes called artificial neurons. Each neuron receives the weighted input of the previous level, and calculates a nonlinear value (activation signal) with the activation function as Figure 2.1 shown. The neuron usually has a threshold. When the activation signal exceeds the threshold, it will send the signal to the next layer. This structure enables the neural network to approximate any nonlinear function in theory, so as to fit the behavior of various complex systems.

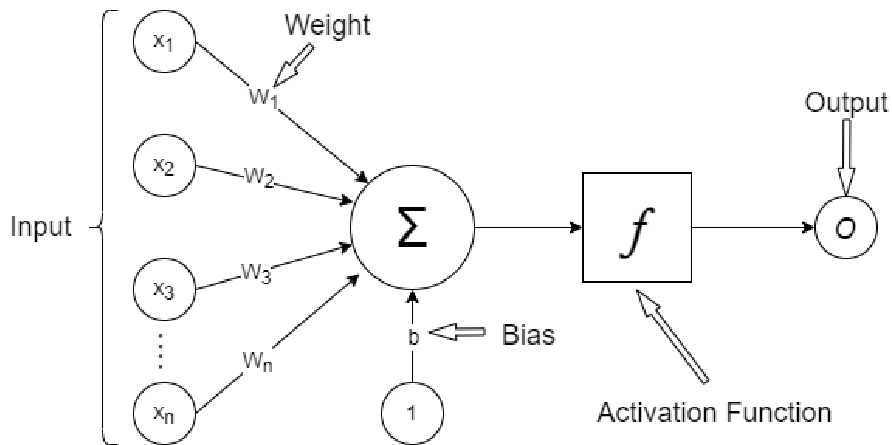


Figure 2.1: Schematic diagram of artificial neuron structure

The calculation equation of weighted input received by a single neuron can be indicated as:

$$Net = \sum_{i=1}^{n} x_i w_i + b \tag{2.1}$$

in (2.1), Net represents the weighted sum of various inputs connected with the neuron, x_i and w_i represent the input value and corresponding weight respectively, b

represents the bias of the neuron, and n represents the number of input dimensions connected with the neuron. After Net is obtained, it is brought into a function to calculate the output value. This function called activation function. The commonly used activation functions include Sigmoid function, Tanh function and Relu function.The activation function introduces nonlinear factors into neurons, which changes the neural network from a linear regression model to a nonlinear model, so that it can learn the nonlinear complex arbitrary function mapping between input and output, and can approach any nonlinear function arbitrarily in theory. The function can be expressed as:

$$f(Net) = f(\sum_{i=1}^{n} x_i w_i + b) \tag{2.2}$$

2.2 Fully-Connected Neural Network(FCN)

A Fully connected neural network is the most common network structure of deep learning. There are three basic types of layers: input layer, hidden layer and output layer. Each neuron in the network can receive the signal from the previous/later layer, and can also feed back the signal to the previous/later layer. Therefore, according to the transmission direction of the signal, the calculation process is divided into two categories: forward propagation and backward propagation.

Forward propagation is the process of inputting a sample vector to the network. The elements of the sample vector are nonlinearly activated by the step-by-step weighted sum of each hidden layer, and finally a prediction vector is output by the output layer. In the process that the input passes through the hidden layer to obtain the output, the output can be regarded as the composite function of the input. When calculating the gradient of the error between the output and the real value to the neurons of layer K, according to the chain rule, the error gradient of neurons close to the output is calculated first, and then the gradient inward is calculated layer by layer, just like the derivative of the composite function, first calculate the derivative of the outer layer, and then the derivative of the inner layer. This process is called back propagation. The following subsections will give a detailed theoretical derivation of FCN model.

2.2.1 Forward propagation derivation

A neural network model can be built with multi-dimensional input, multi-dimensional output, multi-layer and fully connected structure, as shown in the Figure 2.2.

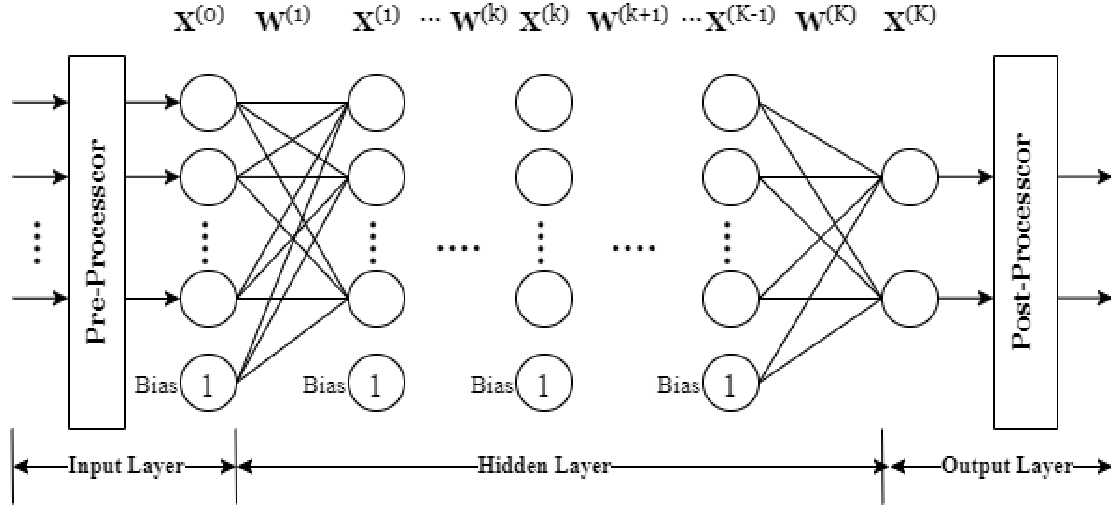


Figure 2.2: FCN model structure diagram

Suppose that the number of layers of the neural network is $K(K>1)$, and the number of neurons in each layer from the input layer to the output layer (excluding bias nodes) are n_0, n_1, n_2,..., n_K, which defines the dimension of the input vector as n_0 and the dimension of the output vector as n_K. The output vector of each layer of the network is expressed as follows:

$$\text{Input Layer: } X^0 = [X_1^{(0)}, X_2^{(0)}, ..., X_{n_0}^{(0)}]^T$$
$$\text{Hidden Layer 1: } X^0 = [X_1^{(1)}, X_2^{(1)}, ..., X_{n_1}^{(1)}]^T$$
$$\text{Hidden Layer 2: } X^0 = [X_1^{(2)}, X_2^{(2)}, ..., X_{n_2}^{(2)}]^T$$
$$\dots$$
$$\text{Output Layer: } X^K = [X_1^{(K)}, X_2^{(K)}, ..., X_{n_K}^{(K)}]^T$$

then define the weight matrix and bias vector of each layer:

$$W^1 \in \mathbb{R}^{n_1 \times n_0} \qquad b^1 \in \mathbb{R}^{n_1 \times 1}$$
$$W^2 \in \mathbb{R}^{n_2 \times n_1} \qquad b^1 \in \mathbb{R}^{n_2 \times 1}$$
$$\dots$$
$$W^2 \in \mathbb{R}^{n_K \times n_K - 1} \qquad b^1 \in \mathbb{R}^{n_K \times 1}$$

Define the activation function of each layer as $f(1)$, $f(2)$,..., $f(K)$; Generally, the corresponding activation function is selected according to different applications. The type of activation function used in each layer can be the same or different. Next, by deriving the output vector expression of hidden layer 1, the general expression of all layers except the input layer can be obtained:

$$net_i^{(1)} = \sum_{j=1}^{n_0} W_{i,j}^{(1)} X_j^{(0)} + b_i^{(1)}, (1 \le i \le n_1)$$

$$net^{(1)} = W^{(1)} X^{(0)} + b^{(1)}$$

$$net^{(1)} = [net_1^{(1)}, net_2^{(1)}, \dots, net_{n_1}^{(1)}]^T$$

$$X^{(1)} = f^{(1)}(net^{(1)}) = [X_1^{(1)}, X_2^{(1)}, \dots, X_{n_1}^{(1)}]^T$$

In the above equations, each element in $net^{(1)}$ represents the weighted sum of the input layer vector and the bias vector, and can also become the input vector of

the first hidden layer. Thus, for layer K ($k \in \{1, 2, \ldots, K\}$):

$$net_i^{(K)} = \sum_{j=1}^{n_{k-1}} W_{i,j}^{(k)} X_j^{(k-1)} + b_i^{(k)}, (1 \le i \le n_k)$$

$$net^{(k)} = W^{(k)} X^{(k-1)} + b^{(k)}$$

$$net^{(k)} = [net_1^{(k)}, net_2^{(k)}, \ldots, net_{n_k}^{(k)}]^T$$

$$X^{(k)} = f^{(k)}(net^{(k)}) = [X_1^{(k)}, X_2^{(k)}, \ldots, X_{n_k}^{(k)}]^T$$

Through forward layer by layer calculation, the $net^{(k)}$ and $X^{(k)}$ of each layer in the neural network can be obtained, that is, the input vector and output vector of each layer, so as to obtain the input value and output value of each neuron. This process is the forward propagation of the neural network.

2.2.2 Backward propagation derivation

This is a back-propagation behavior dominated by error, which aims to obtain the optimal global parameter matrix. The input signal is transmitted forward until the output produces error, and the back-propagation error information updates the weight matrix. Therefore, the model minimizes the error between the output and the target by adjusting the weight W of each neuron in the network. The error between the output value of the neural network and the actual value is calculated by the loss function, which is defined as the mean square error of the output vector, as shown below:

$$L = \frac{1}{2} \sum_{i=1}^{n_k} (X_i^K - T_i)^2 = \frac{1}{2} \sum_{i=1}^{n_k} (\delta_i)^2 \tag{2.3}$$

The purpose of training the neural network model is to obtain a set of network weights W to minimize the value of the loss function. The most basic method is to make the derivative of the loss function to W equal to 0, that is $\frac{\delta L}{\delta W} = 0$. However, for complex networks, it is difficult to solve them analytically. Therefore, in the field of machine learning, the method called Stochastic Gradient Descent(SGD) is widely used for numerical solution, and its expression is as:

$$W \leftarrow W - \eta \frac{\partial L}{\partial W} \tag{2.4}$$

where η is the learning rate of FCN, i.e. the step of searching the weight value, and $\frac{\partial L}{\partial W}$ is the gradient. However, (2.4) is only an abstract expression and cannot be directly used to iteratively solve the weights of each layer in the network. Therefore, it is necessary to adopt the error back propagation method to calculate the gradient layer by layer start from the output layer and update the weights layer by layer.

For $k \leftarrow K, k-1, ..., 1$, the goal is to calculate $\frac{\partial L}{\partial W^{(k)}}$ and $\frac{\partial L}{\partial b^{(k)}}$ for each layer, which can be obtained from (2.4):

$$W^{(k)} \leftarrow W^{(k)} - \eta \frac{\partial L}{\partial W^{(k)}}$$

$$b^{(k)} \leftarrow b^{(k)} - \eta \frac{\partial L}{\partial b^{(k)}}$$

Since the error of the hidden layer does not exist, gradient descent cannot be directly applied to it, but the error is first propagated back to the hidden layer, and

then gradient descent is applied, in which the process of passing the error forward from the last layer uses a method called chain rule. For the K layer, the partial derivative of the loss function to a certain weight of the layer can be expressed as the continuous multiplication of three partial derivatives, expressed as:

$$\begin{cases} \frac{\partial L}{\partial W_{i,j}^{(k)}} = \frac{\partial L}{\partial X_i^{(k)}} \frac{\partial X_i^{(k)}}{\partial net_i^{(k)}} \frac{\partial net_i^{(k)}}{\partial W_{i,j}^{(k)}} = S_i^{(k)} \frac{\partial net_i^{(k)}}{\partial W_{i,j}^{(k)}} \\ \frac{\partial L}{\partial b_{i,j}^{(k)}} = \frac{\partial L}{\partial X_i^{(k)}} \frac{\partial X_i^{(k)}}{\partial net_i^{(k)}} \frac{\partial net_i^{(k)}}{\partial b_{i,j}^{(k)}} = S_i^{(k)} \frac{\partial net_i^{(k)}}{\partial b_{i,j}^{(k)}} \\ i \in \{1, 2, ..., n_k\} \quad j \in \{1, 2, ..., n_{k-1}\} \end{cases} \tag{2.5}$$

In (2.5), $S_i^{(k)}$ represents the product of the first two partial derivatives, i.e,

$$S_i^{(k)} = \frac{\partial L}{\partial X_i^{(k)}} \frac{\partial X_i^{(k)}}{\partial net_i^{(k)}} \tag{2.6}$$

Further expand the partial derivatives of (2.5) to obtain the following equations:

$$\frac{\partial L}{\partial X_i^{(k)}} = \begin{cases} (X_i^{(K)} - T_i) = \delta_i, k = K \\ \sum_{l=1}^{n_{k+1}} \frac{\partial L}{\partial net_l^{(k+1)}} \frac{\partial net_l^{(k+1)}}{\partial X_i^{(k)}} = \sum_{l=1}^{n_{k+1}} \frac{\partial L}{\partial net_l^{(k+1)}} W_{l,i}^{(k+1)}, k \neq \text{K} \end{cases} \tag{2.7}$$

$$\frac{\partial X_i^{(K)}}{\partial net_i^{(k)}} = f'^{(k)}(net_i^{(k)}) \tag{2.8}$$

$$\frac{\partial net_i^{(k)}}{\partial W_{i,j}^{(k)}} = X_j^{(k-1)} \tag{2.9}$$

$$\frac{\partial net_i^{(k)}}{\partial b_i^{(k)}} = 1 \tag{2.10}$$

According to (2.5) to (2.10), starting from the output layer, $\frac{\partial L}{\partial W_{i,j}^{(k)}}$ and $\frac{\partial L}{\partial b_i^{(k)}}$ and the matrix vectors they represent are deduced layer by layer. To sum up, we can get the whole process of FCN model updating the network weight once:

1. Forward propagation
   For $k \leftarrow 1, 2, ..., K$:

$$net^{(k)} \leftarrow W^{(k)} X^{(k-1)} + b^{(k)}$$

$$X^{(k)} \leftarrow f^{(k)}(net^{(k)})$$

2. Backward propagation
   For $k \leftarrow K, k-1, ..., 1$:

$$S^{(k)} \leftarrow \begin{cases} f'^{(k)}(net^{(k)}) \odot \delta, k = K \\ f'^{(k)}(net^{(k)}) \odot (W^{(k+1)^T} s^{(k+1)}), k \neq \text{K} \end{cases}$$

$$\frac{\partial L}{\partial W^{(k)}} \leftarrow \text{S}^{(k)} X^{(k-1)^T}$$

$$\frac{\partial L}{\partial b^{(k)}} \leftarrow \text{S}^{(k)}$$

3. Updated network parameters through layers
   For $k \leftarrow K, k-1, ..., 1$:

$$W^{(k)} \leftarrow W^{(k)} - \eta \frac{\partial L}{\partial W^{(k)}}$$

$$b^{(k)} \leftarrow b^{(k)} - \eta \frac{\partial L}{\partial b^{(k)}}$$

2.3 Recurrent Neural Network(RNN)

2.3.1 Original Recurrent Neural Network

Recurrent neural network is an improved model of FCN. In this kind of model, each neuron in the hidden layer has the ability to accept specific time-delay input. When using this kind of neural network, we need to add the information of the last time in the current iteration. For example, when we try to predict the output of the next time step in a time series event, we need to know the output information of the previous time step first. RNN can process input and share any length input and weight across time, while the model size will not increase with the size of the input. The calculation in the model also needs to consider the existing historical information. This is the difference between RNN and FCN. The feedback loop of RNN will be connected to its past prediction and take its own historical output as input, which means that RNN has memory. The purpose of adding memory to the neural network is that the sequence itself contains information. This continuous information is stored in the hidden state of the recurrent network. This hidden state management spans multiple time steps and passes forward layer by layer, affecting the processing of each new sample by the network. This structure makes RNN particularly good at capturing the correlation of events in the sequence, which is called "long-distance dependence", because the events downstream of time depend on one or more previous events and are a function of these events.

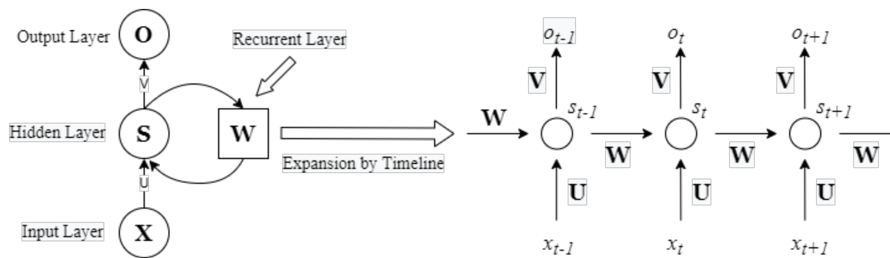


Figure 2.3: RNN structure diagram

The basic structure of RNN is very simple, the output of the network is saved in a memory unit, which enters the neural network together with the next input. A simple two-layer network is used as an example, based on which the structure of RNN is expanded, as shown in Figure 2.3. Where X represents the value of the input layer, S represents the value of the hidden layer, u represents the weight matrix from the input layer to the hidden layer, O is the value of the input layer, and v represents the weight matrix from the hidden layer to the output layer. The special feature of RNN is that the value S of the hidden layer depends not only on

the current input X, but also on the value S of the last time input. The weight matrix W is the weight of the last value of the hidden layer as the input of this time.

By further expanding this simplified structure along the timeline, the structure on the right side of Figure 2.3 can be obtained. In this expansion structure, t represents the order in which RNN receives information, that is, after the network receives the input value x_t at time t, the values of the corresponding hidden layer and output layer are s_t and o_t respectively. The key point is that the value of s_t depends not only on x_t, but also on the hidden layer value s_{t-1} at the previous time. Similarly, s_t at time t will also participate in the calculation of s_{t-1} at the next time. The calculation method of RNN can be expressed as:

$$o_t = g(VS_t) \tag{2.11}$$

$$S_t = f(US_t + WS_{t-1}) \tag{2.12}$$

Since the RNN model came into being, the RNN structure has been improved and modified, resulting in various variants based on RNN, such as Long-Short Term Memory(LSTM) model and Gated Recurrent Unit(GRU) model. Next, the two RNN variant models of LSTM and GRU will be explained in detail.

2.3.2 Long-Short Term Memory model

RNN has two obvious technical disadvantages: gradient vanishing and gradient explosion [5]. Since the prediction error propagates back along each layer of the neural network, when the eigenvalue of the Jaocibian matrix is greater than 1, the gradient of each layer will increase exponentially as it is farther and farther from the output, resulting in gradient explosion; On the contrary, if the eigenvalue of Jaocibian matrix is less than 1, the gradient will decrease exponentially and the production gradient will disappear. For RNN model, the gradient vanishing means that the prediction effect of neural network cannot be improved by deepening the network layer, because in terms of how to deepen the network, only several layers close to the output can really play a role in learning. This makes it difficult for the RNN model to learn the long-distance dependence in the input sequence. The problem of gradient explosion can be alleviated by gradient clipping, that is, when the normal form of the gradient is greater than a given value, the gradient is proportionally contracted. The gradient disappearance problem is relatively difficult, and the model itself needs to be improved. LSTM model makes up for the loss caused by gradient vanishing to a great extent by adding gating mechanism [6].

The structure of LSTM is shown in Figure 2.4. Compared with RNN, LSTM still calculates value h_t based on the input x_t at time t and the memory h_{t-1} at time $t-1$. However, LSTM internal structure is designed more finely, adding three gate mechanisms of input gate i_t, forget gate f_t and output gate o_t and an internal memory unit c_t. The input gate controls the new state of the current calculation to be updated to the memory unit to a certain extent; The forget gate controls how much information in the front step memory unit is forgotten; The output gate controls how much the current output depends on the current memory unit. The iteration of each part of LSTM is further explained below.

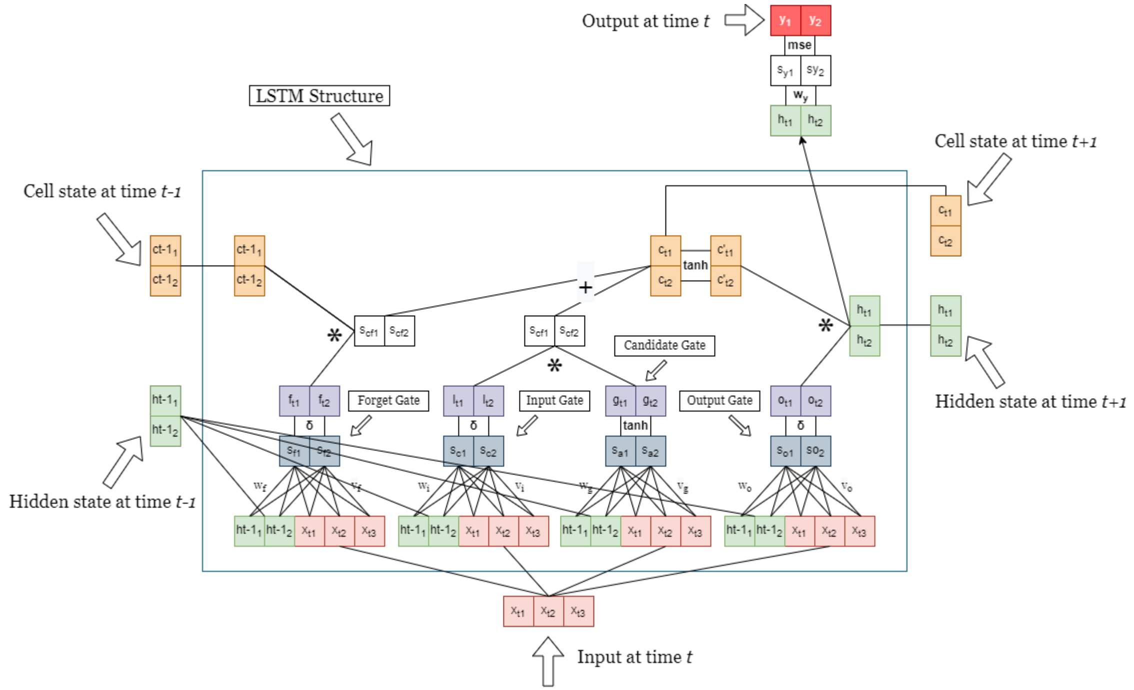


Figure 2.4: LSTM structure diagram

1. Forget gate
   The first step of LSTM is to decide what information to discard from the cell state. This decision is made by the so-called "forget gate" network layer like Figure 2.5 shown. It receives h_{t-1} and x_t and calculates f_t by Sigmoid function. f_t represents the forget rate, and f_t is between 0 and 1 for each value of cell state c_{t-1}. 1 means "fully accept this" and 0 means "completely ignore this". The forget gate adopts the calculation formula as:

$$f_t = \sigma(w_f h_{t-1} + v_f x_t) \tag{2.13}$$

2. Input gate
   This step is to determine what new information needs to be stored in the cell state. There are two parts here shown in Figure 2.6. In the first part, a so-called "input gate" network layer determines which information needs to be updated. In the second part, a candidate network layer creates a new candidate value vector c_t by tanh function, which can be used to add to the cell state. In the next step, we combine the above two parts to produce the update of cell state. The input gate adopts the calculation formulas as:

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.14}$$

$$g_t = tanh(w_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.15}$$

3. Cell state update
   Now update the old cell state c_{t-1} to c_t. The required values have been determined in the previous steps. In this step, the cell state is updated according to

$$C_t = f_t C_{t-1} + i_t g_t \tag{2.16}$$

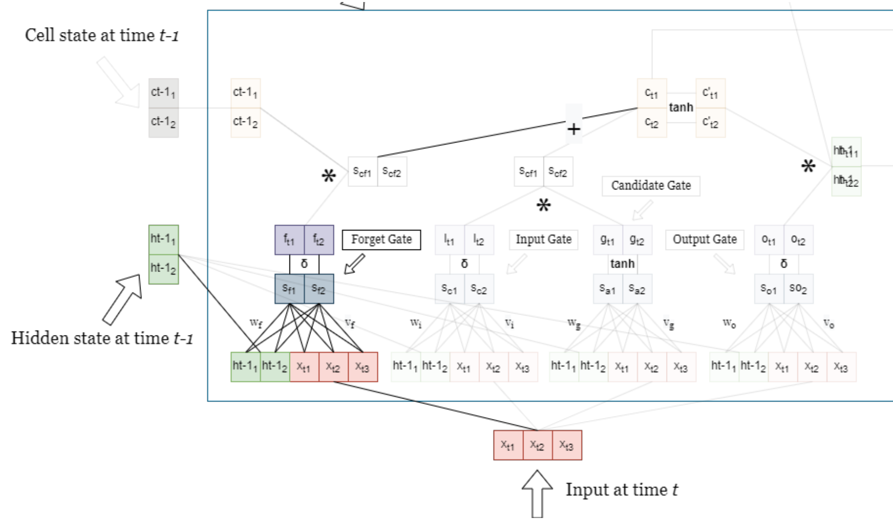


Figure 2.5: Forget gate step diagram

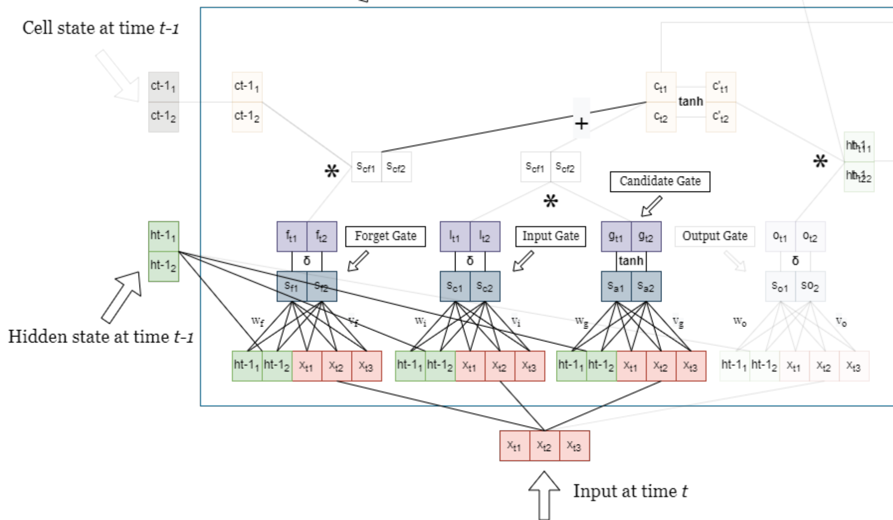


Figure 2.6: Input gate step diagram

The cell state at time $t-1$ is multiplied by f_t to forget what the forget gate decides to forget, and then the new candidate value g_t provided by the candidate gate is added and scaled according to the value i_t determined by the input gate.

4. Output gate
   Finally, we need to determine the output value. The output depends on the cell state, but it will be a "filtered" value, the procedure shown in Figure 2.8. Firstly, the output proportional value o_t of C_t is determined after Sigmoid activation function. Then, c_t is calculated by tanh function (adjust the value between -1 and 1) and multiplied by o_t, so that hidden state value h_t can be determined. The calculation equations of o_t and h_t are expressed:

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{2.17}$$

$$h_t = o_t tanh(C_t) \tag{2.18}$$

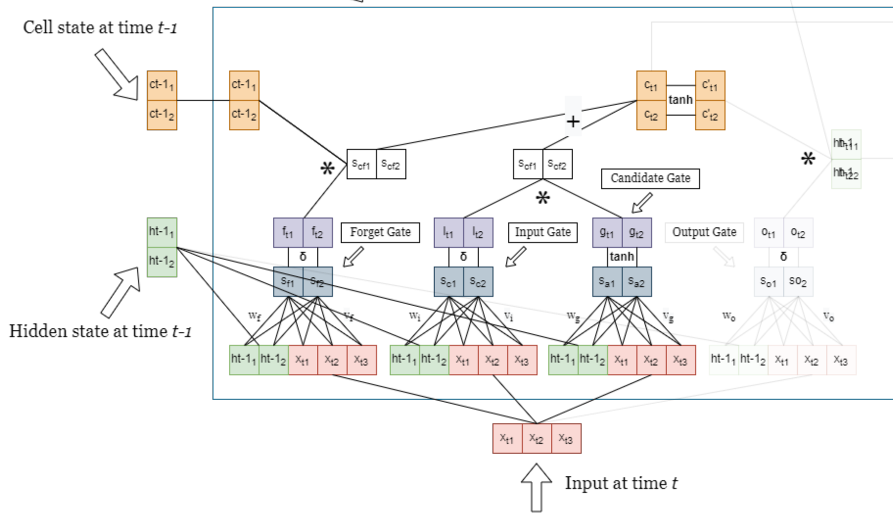


Figure 2.7: Cell state update step diagram

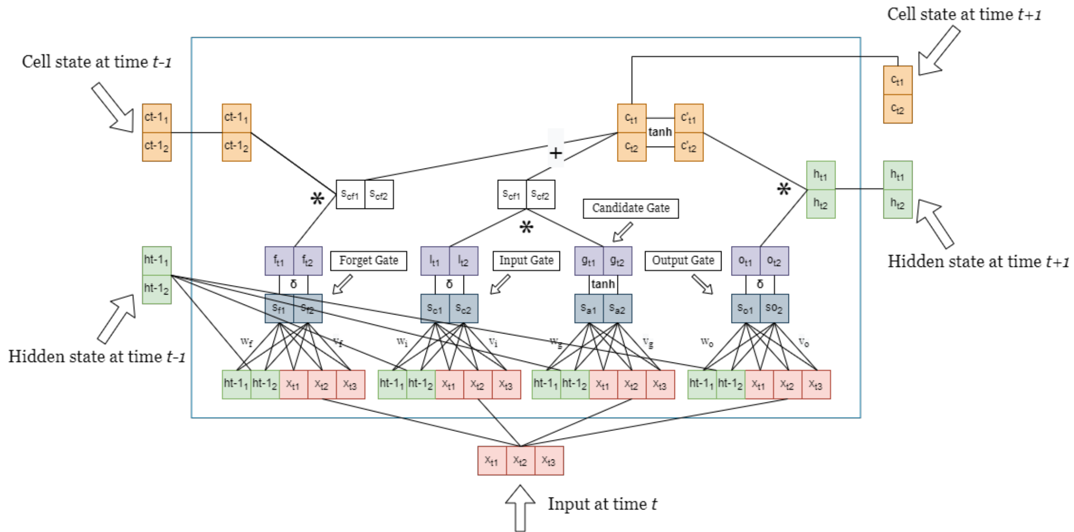


Figure 2.8: Output gate step diagram

2.3.3 Gated Recurrent Unit model

Another RNN variant is called GRU [7]. LSTM controls the input value, memory value and output value respectively through three gate functions: input gate, forget gate and output gate. There are only two gates in GRU: update gate z_t and reset gate r_t. The update gate is used to control the extent to which the state information of the previous time is brought into the current state. The larger the value of the update gate, the more the state information of the previous time is brought in; The reset gate controls how much information of the previous state is written to the current candidate set h_t. The smaller the value of the reset gate, the less information of the previous state is written. This makes GRU simpler than the standard LSTM model. The basic structure of GRU model is shown in Figure 2.9.

1. Reset gate and update gate

   The input-output structure of GRU is the same as that of RNN. There is a

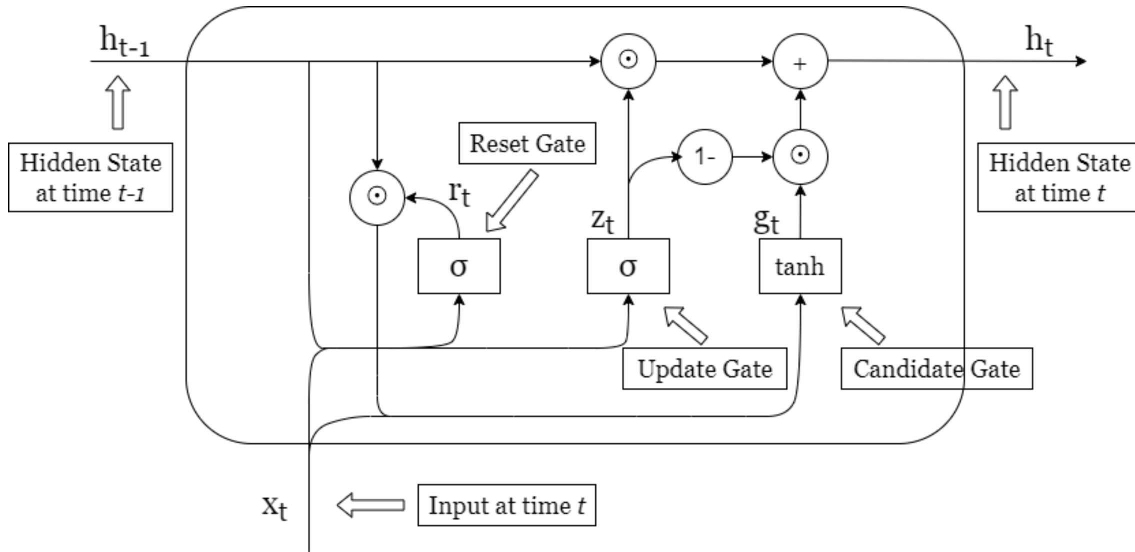


Figure 2.9: GRU structure diagram

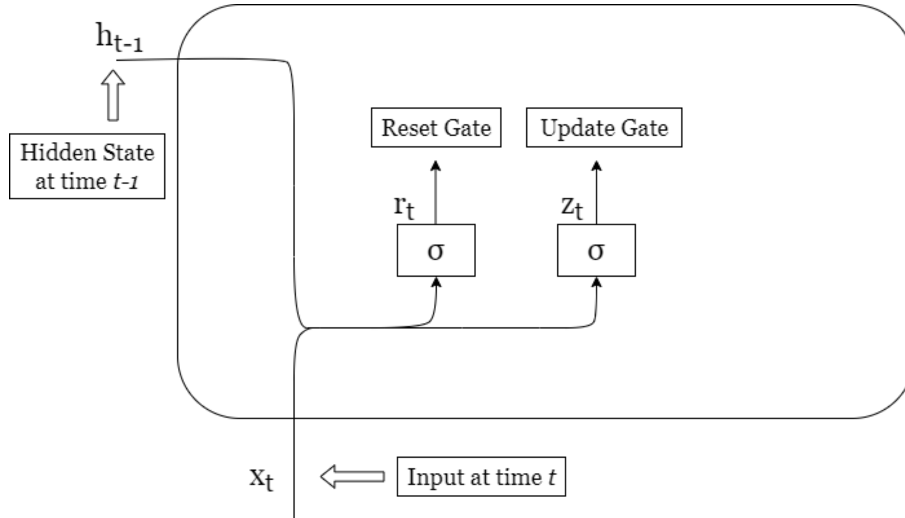


Figure 2.10: Forget and reset gate step diagram

current input x_t and a hidden state h_{t-1} passed down from the previous step like Figure 2.10 shown. The hidden state contains the relevant information of the previous step. The combined values of x_t and h_{t-1} get two gating States after the reset door and the update door respectively. The calculation equations of reset door and update door are expressed as:

$$r_t = \sigma(x_t w_{xr} + h_{t-1} w_{hr} + b_r) \tag{2.19}$$

$$z_t = \sigma(x_t w_{xz} + h_{t-1} w_{hz} + b_z) \tag{2.20}$$

2. The candidate hidden state

$$g_t = tanh(x_t w_{xh} + (r_t \odot h_{t-1}) w_{hh} + b_h) \tag{2.21}$$

The calculation equation of candidate hidden states is (2.21). Here $x_t w_{xh}$

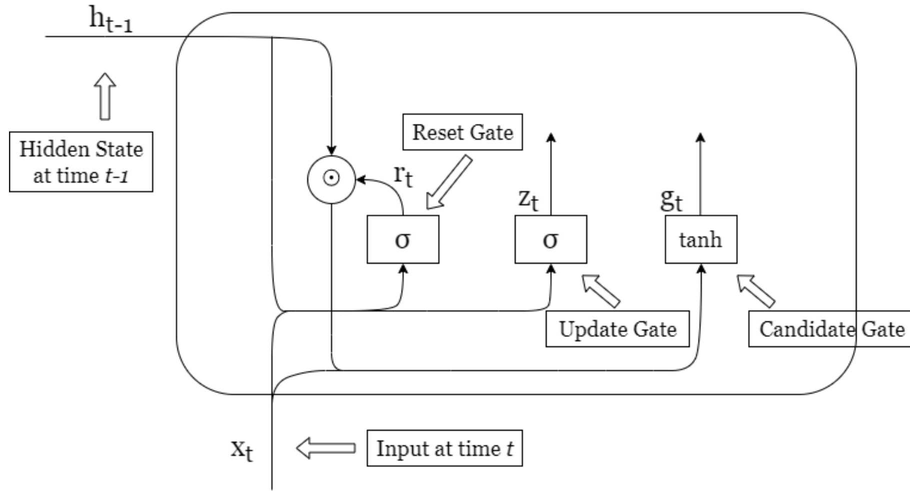


Figure 2.11: Candidate hidden state step diagram

represents the information of the current time, r_t represents how much information of the past that should be retained: when r_t is close to 0, it means to forget the hidden state of the previous time, i.e. reset; When r_t is close to 1, it means that the hidden state of the previous time is completely retained. The detailed unit diagram is shown in Figure 2.11.

3. The update hidden state

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot g_t \tag{2.22}$$

The last step is to calculate the hidden state at the current time. The calculation equation of this step was shown in (2.22). When z_t approaches 1, GRU tends to output the historical hidden state value h_{t-1}; On the contrary, when z_t is close to 0, GRU tends to output the candidate hidden state value g_t. The complete steps thus far are shown in Figure 2.9.

2.4 Transformer

2.4.1 Transformer introduction

At present, a widely used model is the Seq2Seq model based on encoder-decoder architecture, that is, the input sequence is first processed into context sequence by an encoder, and then processed into a target sequence by the decoder. In the Seq2Seq model, the encoder and decoder generally adopt the RNN structure, such as the standard RNN and LSTM. Referring to the principle explanation of RNN in the previous chapter, we can know that RNN can capture the dependencies between front and rear sequences or elements in different ranges. In order to overcome the problem of information inundation caused by long sequences, the attention mechanism is introduced into the standard encoder-decoder architecture. By giving different attention weights to different elements in the front and back sequences, the important elements in the sequence are projected more attention, so that the important information is not inundated in the long sequence. However, there is a serious disadvantage of using the RNN structure, that is, RNN belongs to sequence model. The input information needs to be processed one by one in series, and the attention weight needs to be determined after all the sequences are input into the model. Therefore, it requires a lot of time overhead in both training stage and inference stage, so it is difficult to realize parallel processing.

In 2017, Google Research Institute and other teams jointly proposed a new attention model to replace the previous RNN structure Seq2Seq model. This model can obtain all inputs at one time and use the attention mechanism to combine the input weights of different distances. This new model is called transformer [8], and its structure is shown in Figure 2.12.

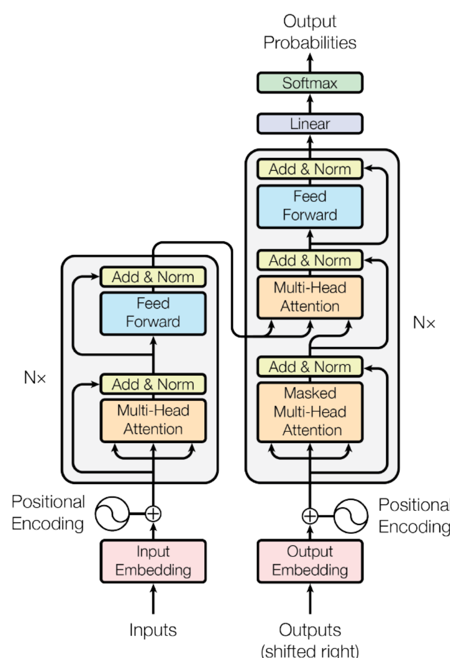


Figure 2.12: Transformer structure diagram[8]

the transformer also adopts encoder-decoder architecture, but in this model, encoder and decoder no longer use RNN structure, but replace "encoder stack" and "decoder stack". Encoder stack and decoder stack respectively represent six consecutive encoders and decoders with the same structure, as shown in Figure 2.13.

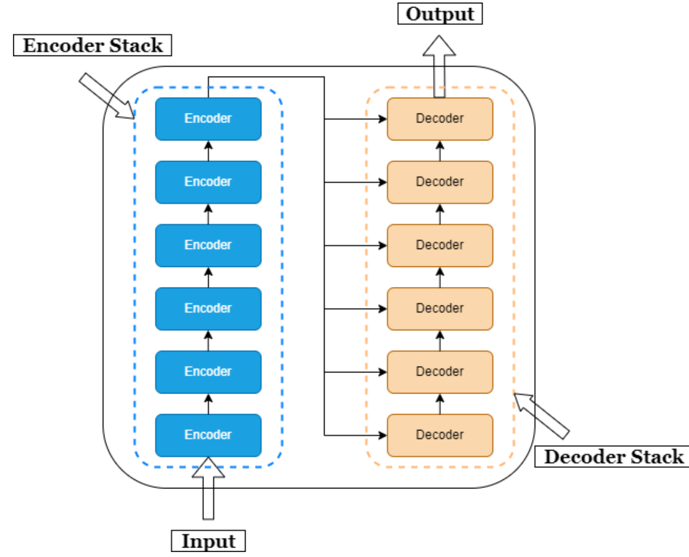


Figure 2.13: Encoder stack and decoder stack diagram

Each decoder contains two sub networks in series: a self-attention module (specifically called multi-head attention module) and a feedforward neural network. Each sub network has a residual connection, and its output is determined as:

$$O_{rc} = LayerNorm(Sublayer(x) + x) \tag{2.23}$$

in this equation, $Sublayer(x)$ represents the specific mapping operation of the sub network to the input feature x, x is the identity mapping, and $LayerNorm(\cdot)$ is the feature normalization operation. For the decoder, in addition to the self attention module and fully connected feedforward network similar to the decoder, another attention module is added between the two sub networks (This attention module is called "coding-decoding attention" module, which also adopts multi-head attention structure). Similar to the encoder, the three sub networks in the decoder also have residual connections, and normalization is performed after each residual synthesis. The specific structure of a single decoder and encoder is shown in Figure 2.14.

2.4.2 Principle of attention mechanism

The attention mechanism can be described as a function that maps a query(abbreviated as Q) and a key-value set(abbreviated as K and V respectively, this key value set is also known as "source") with M elements to a value. This function can be expressed as:

$$attention : Q, \{k_i, v_i\}_M \longrightarrow V \tag{2.24}$$

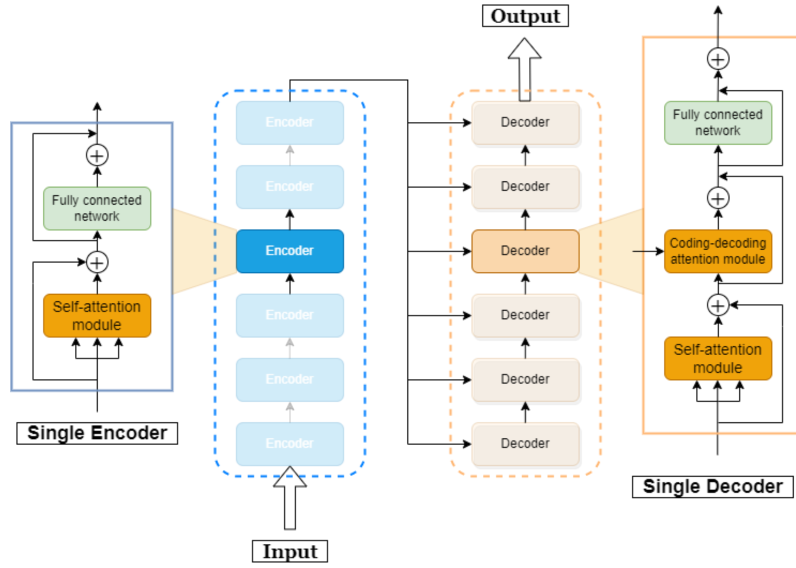


Figure 2.14: Single encoder and decoder structure diagram

The calculation of attention can be divided into three steps:

1. Using the input Q and each $k_i(i = 1, 2, ..., M)$ in the set $\{k_i, v_i\}_M$ to calculate the similarity,

$$similarly(Q, \{k_i, v_i\}_M) = (similarly(Q, k_1), ..., similarly(Q, k_M)) \quad (2.25)$$

where M kinds of similarity are obtained.

2. M similarity degrees are probabilistic through the softmax function

$$[p_1, ..., p_M] = softmax(Q, \{k_i, v_i\}_M) \quad (2.26)$$

and a M-dimensional probability distribution will be obtained in this step.

3. Using the probabilistic similarity as the weight coefficient, the v_i in the set $\{k_i, v_i\}_M$ is weighted and summed by

$$V = attentionQ, \{k_i, v_i\}_M = \sum_{i=1}^{M} p_i v_i \quad (2.27)$$

to obtain the final output V.

In the above equations,$similarly(\cdot, \cdot)$ is a similarity function, and one of the simplest implementation forms is to calculate the dot product of the two input vectors. From the perspective of mechanism, the focusing process of attention mechanism is reflected in the weight coefficient. The greater the weight, the more attention is projected on the corresponding value, that is, the weight represents the importance of information. In the transformer model, two kinds of attention realization methods are given: scaled dot-product attention(SDPA) and multi-head attention (MHA). And SDPA is expressed as:

$$attention_{SDPA}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (2.28)$$

where d_k is the dimension of key (i.e. query). When d_k is large, the dot product result will become very discrete. Therefore, divide the dot product result by $\sqrt{d_k}$ to normalize it. In (2.28), Q is the query matrix (the matrix form composed of multiple query vectors), and K and V are the keys and values represented by the matrix respectively. Figure 2.15 shows the data flow diagram and matrix calculation example of SDPA.

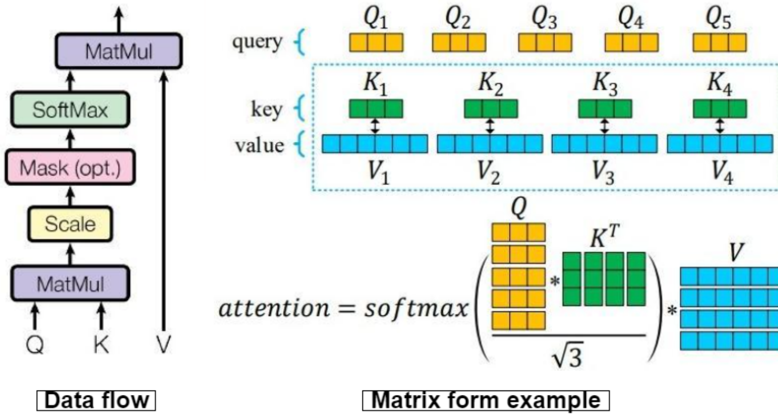


Figure 2.15: Data flow diagram and matrix calculation example of SDPA

MHA can be considered as multi-channel fusion SDPA. Its basic operation process includes four steps: multi-channel linear transformation(2.29), multi-channel SDPA(2.30), multi-channel fusion(2.31) and re-linear transformation(2.32). Let the vector dimensions of a key (i.e. query) and value be d_k and d_v respectively, let the number of rows of query matrix Q (i.e. the number of query entries) be M_Q, and the number of rows of key matrix K (the same number of rows with value matrix V, i.e. the number of elements of key-value set) be M_K, then $W_i^Q \in \mathbb{R}^{M_q \times d_k}$, $W_i^K \in \mathbb{R}^{M_k \times d_k}$, $W_i^V \in \mathbb{R}^{M_k \times d_v}$ and $W_i^O \in \mathbb{R}^{hd_v \times M_q}$ in the above equations are learnable linear transformation parameters; $i = 1, 2, ..., h$ is the "number of heads" of MHA (set $h = 8$ in transformer model).

$$Q_i^{'} = QW_i^Q, K_{'}^Q = KW_i^K, V_i^{'} = VW_i^V \tag{2.29}$$

$$head_i = attention_{SDPA}(Q_i^{'}, K_i^{'}, V_i^{'}) \tag{2.30}$$

$$head_f = concat_{(}head_1, ..., head_h) \tag{2.31}$$

$$attention_{MHA}(Q, K, V) = head_f W^O \tag{2.32}$$

2.4.3 Principle of self-attention mechanism

Three sets are involved in the description of the attention mechanism: query set Q, key set K and value set V. However, these sets do not have to be different. There are three situations:

1. If $Q \neq K \neq V$, that is, the three sets are completely different (this is the case in the above example of SDPA and MHA), this situation is referred to as QKV mode for short.

2. If $Q \neq K = V$, that is, the key and value sets are the same set, this situation is referred to as QVV mode for short.
3. If $Q = K = V$, that is, the three sets of query, key and value are actually equal, this situation is referred to as VVV mode for short.

The attention model uses other things as a query. For example, the attention used in RNN is to take the output of the previous time as the query and calculate the correlation between the current input (equivalent to weighting the historical information generated by the previous input and the current input to determine which information that is more important). In the VVV mode mention above, it is precise since this mode does not need external information and directly calculates the interaction relationship between the features of the same layer. For example, if you input a sequence of information, you can directly calculate the correlation between the information under the sequence, that is, Q, K and V matrices are all generated by the sequence information itself, so attention in this mode is called "self attention".

Among the above three attention modes, QVV and VVV are the most widely used, because they contain two very important problems in the feature representation: the QVV mode represents how to use one feature set to represent another set, while the VVV mode represents how to use one feature set to represent itself. In the transformer model, the attention mechanisms of the above two models are involved. There are three types of attention modules in the transformer model: encoder self-attention module (belonging to the VVV mode), decoder self-attention module (belonging to the VVV mode) and encoding-decoding attention module (belonging to the QVV mode), they are discussed below.

2.4.3.1 Encoder self-attention module:

The self-attention module adopts the MHA structure and belongs to the VVV mode, where V is the coding of each dimension input in the input sequence by the previous encoder (the coding here can be understood as the hidden state vector in RNN, that is, internal representation of each dimension vector in the input. If it is the first encoder, the coding here is the embedded vector of each input dimension). The encoder self-attention module is used to capture the relationship between each dimension vector of the input sequence, such as translating the sentence: "The manager dismissed the employee because of his bad performance", in which "his" refers to manager or employee? The self-attention module of the encoder is to make "employee" to have higher influence on "his", that is, attention weight. Figure 2.16 is a schematic diagram of a self-attention module for the above translation problems, in which $x_1, ..., x_{11}$ respectively represent the coding of each word in the sentence, $y_1, ..., y_{11}$ respectively represent the output of each word by the self-attention module, and $e_{i,j}$ is the attention weight projected on the input x_j when the self attention model outputs y_i. Figure 2.16 takes y_8 (i.e. for the word "his") as an example to illustrate the distribution of attention weight on each input code.

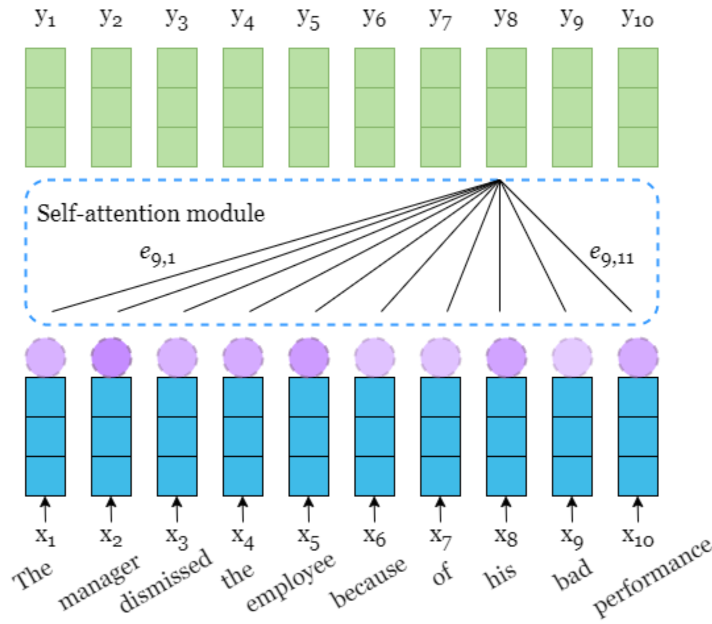


Figure 2.16: Schematic diagram of attention weight distribution

2.4.3.2 Decoder self-attention module

The structure of decoder self-attention module is very similar to that of the encoder. The only difference is that the mask mechanism is added. This is because in the decoder, the self attention module is only allowed to process those items before the current item, which is different from all items that the encoder needs to "see". The realization of the above goal is very simple. As long as all items after the current processing item are hidden with a mask before the attention weight of softmax is probabilistic, the calculation of attention is changed to the form with a mask determined by.

$$attention_{SPDA}(Q, K, V) = softmax(\frac{QK^T \odot M}{\sqrt{d_k}})V \tag{2.33}$$

In (2.33), M is a lower triangular matrix of order M_q, and its non-zero matrices are all 1. Figure 2.17 takes the output y_8 (corresponding to "his" once) as an example to illustrate the execution method of the decoder self-attention module.

2.4.3.3 Encoding-decoding attention module

In a decoder, the encoding-decoding attention module is located behind the self-attention module of the decoder. The module is also constructed as an MHA structure, but belongs to the QVV mode. Where Q comes from the output of the previous decoder and V comes from the output of the last encoder (that is, the final output of the encoder stack). The attention module enables each item in the decoder to scan each dimension of the input sequence with weight.

The attention module is followed by a feedforward neural network layer, which is constructed as a fully connected network operated item by item, and the output

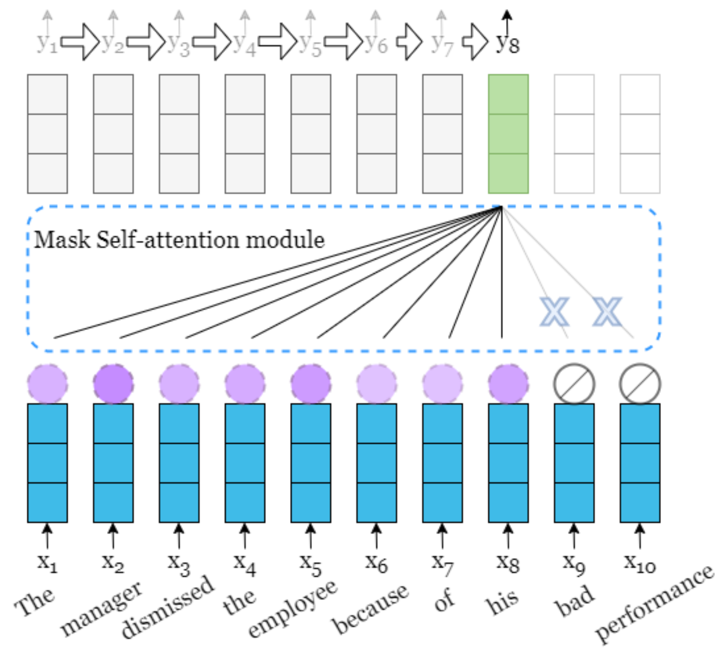


Figure 2.17: Schematic diagram of attention weight distribution

vector obtained by each attention module is processed independently. After the last decoder processes the features, the obtained results are then sent to a linear transformation layer and a softmax layer, in which the linear transformation layer transforms the dimension of the decoder output vector to the standard length, which is similar to the initial word embedding operation. Then the vector is probabilistic by the subsequent softmax layer, and the result is the probability distribution of each input on the target resource.

3

Neural network training process and strategy

3.1 Determine the hyper-parameter of neural network

Hyper-parameters refer to the parameters that need to be set before the neural network starts training, rather than the parameters obtained by the network through learning. Hyper-parameters affect the learning speed and final prediction results of the neural network and usually manually adjusted by experience, which requires the trainer to have some training experience or even intuition, and can judge the influence of a group of hyper-parameters based on the training results, so as to optimize the hyper-parameters to improve the learning performance and prediction effect. The main hyper-parameters in the neural network are listed in Table 3.1. Next, I will discuss the experience of adjusting hyper-parameters in practical training.

Table 3.1: List of hyper-parameters

Number	Name of hyper-parameter
1	Learning rate
2	Regularization parameters
3	Layers of neural network
4	Number of neurons in each hidden layer
5	Number of learning rounds *epoch*
6	Minibatch size of small batch data
7	Coding mode of output neurons
8	Selection of cost function
9	Weight initialization method
10	Types of neuronal activation functions
11	The scale of the training model data includes

3.1.1 Input data normalization

When solving some practical problems, the sample data we obtain often contains multiple dimensions, that is, a sample is represented by multiple features, and there may be large scale differences in different features of the sample data. Taking this project as an example, the component rotating speed and vehicle speed are the inputs

to a neural network, but the difference between them is three orders of magnitude. Data normalization is to limit the data to a certain range after being processed by some algorithm, so as to eliminate the dimensional influence between various input data. Normalization makes the following data processing more convenient, it has two advantages:

1. Normalization improves the speed of solving the optimal solution by gradient descent method
   From section 2.2.1, we know that the training speed of a neural network is determined by the speed of the gradient descent method. As long as the gradient descent speed becomes faster, the training speed of the neural network will become faster accordingly. Therefore, in order to explain this problem, we analyze this problem from the perspective of gradient descent of the loss function. If our sample has only two features $X1$ and $X2$, the two feature scales differ greatly. Assuming that the feature $X1$ interval is [0,10000] and $X2$ interval is [0,10], we do not normalize them. We design a neural network algorithm for these two features, and then we draw the image of its loss function, as shown in Figure 3.1.

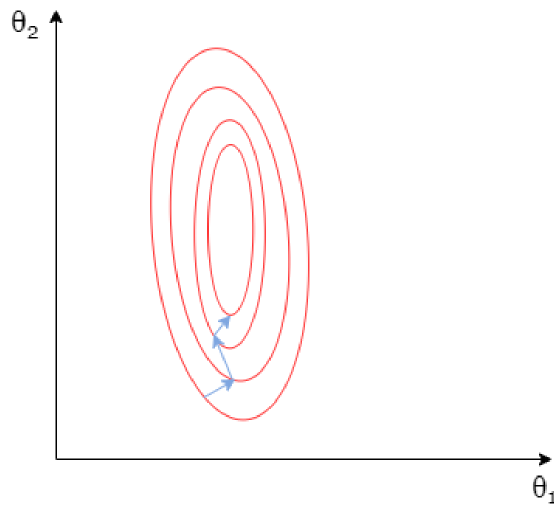


Figure 3.1: Non-normalized input loss gradient descent diagram

   The red circle represents the contour of the two features. Because the interval difference between the two features $X1$ and $X2$ is very large, the contour line formed by them is a long oval and very sharp. The process of using the gradient descent method to find the optimal solution is to find the parameters that minimize the value of the loss function $\theta 1$, $\theta 2$. When the data is not normalized, the gradient descent is likely to take the Z-shaped route (it will go perpendicular to the contour line, because the direction of the gradient is perpendicular to the tangent direction of the contour line). Because the gradient descent route fluctuates greatly, it needs many iterations to converge. This is an intuitive manifestation of the slow training speed of the neural network with non normalized operation.

   After the data is normalized, the coefficient difference in front of the loss function variable θ is small. In this way, the contour surface of the image

is approximately circular. At this time, the gradient direction obtained by the solution can point directly to the center of the circle. When the gradient decreases, it can converge faster and the efficiency is higher. In this way, the training speed of the neural network will be accelerated accordingly. Now we normalize the training set data, and the image of the loss function is shown in Figure 3.1.

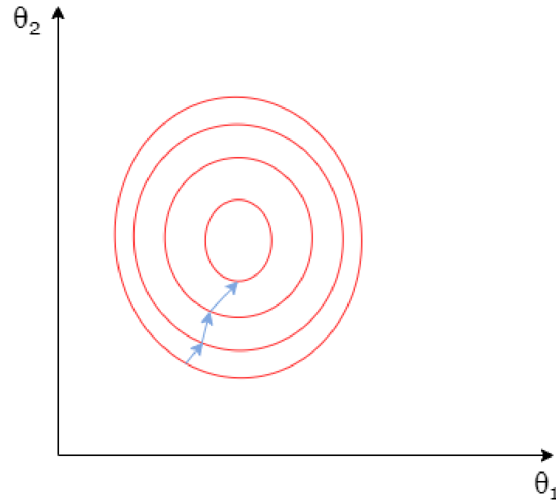


Figure 3.2: Normalized input loss gradient descent diagram

2. Normalization has the potential to improve accuracy
   Some classifiers need to calculate the distance between samples (such as Euclidean distance). If a feature range is very large, the distance calculation depends on this feature. If the actual situation is that the feature with small range is more important, then normalization will play an important role.

In the normalization process of the input data of this project, the linear normalization method is adopted, that is, the linear transformation is carried out on the original data, so that the result value is mapped between 0 and 1, and the conversion is found as:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

3.1.2 Method for determining the architecture of neural network

3.1.2.1 Determine the number of hidden layers

It can be seen from section 2.2.1 that the neural network is divided into three parts: input layer, hidden layer and output layer. The number of nodes in the input and output layer is determined by the project requirements, so it is fixed. How to select the number of hidden layers and the number of neurons in each hidden layer greatly affects the performance of the model.

For generally simple data sets, one or two hidden layers are usually enough. However, for complex data sets involving time series or computer vision, additional

layers are required [9]. Single layer neural networks can only be used to represent linear separation functions, that is, very simple problems. For example, two classes in classification problems can be separated neatly by a straight line. The number of hidden layers and the effect of neural network are listed in Table 3.2.

Table 3.2: Effect of different hidden layers

Number of hidden layer	Model effect
1	Can fit any function that contains a continuous mapping from one finite space to another.
2	With appropriate activation function, it can represent any decision boundary with any accuracy, and can fit any smooth mapping with any accuracy.
3	The extra hidden layer can learn complex features (such as some automatic feature Engineering).

The deeper the number of layers, the stronger the ability to fit the function in theory, and the effect will be better [10]. However, in fact, deeper number of layers may lead to the problem of over fitting. At the same time, it will also increase the difficulty of training and make the model difficult to converge. Moreover, for RNN type models, too many layers cannot be used in time series prediction. The increase in the number of layers will lead to an exponential increase in time and memory overhead. Then, you will find that the gradient between layers vanishing, which is the most fatal point. When the number of layers exceeds three, the disappearance of the gradient between layers becomes very obvious. Coupled with the time series model, the update iteration of LSTM layer or GRU layer close to the input layer slows down, the convergence effect and efficiency drop sharply, and it is even very easy to enter the local minimum [11]. Generally, one or two layers of LSTM/GRU are sufficient for timing prediction, because there are many methods to reduce data complexity in input and output, so as to improve training efficiency. Additional, LSTM or GRU mainly solves the problems of RNN long-distance dependence and gradient vanishing in each layer, not between layers. Therefore, from the perspective of model capacity, no more than three hidden layers are the best choice.

3.1.2.2 Determine the number of neurons in hidden layer

Using too few neurons in the hidden layer will lead to underfitting. On the contrary, using too many neurons can also lead to some problems [12]. First, too many neurons in the hidden layer may lead to overfitting. When the neural network has too many nodes (too much information processing ability), the limited amount of information contained in the training set is not enough to train all neurons in the hidden layer, so it will lead to overfitting. Even if the training data contains enough information, too many neurons in the hidden layer will increase the training time, so it is difficult to achieve the expected effect. Obviously, it is very important to select an appropriate number of hidden layer neurons.

In general, it is sufficient to use the same number of neurons for all hidden layers. For some data sets, having a larger first layer followed by a smaller layer will lead to

better performance, because the first layer can learn many low-order features, which can be fed into the subsequent layer to extract higher-order features. It should be noted that adding more layers will achieve greater performance improvement than adding more neurons in each layer. Therefore, it is not suitable to add too many neurons to a hidden layer.

In short, the optimal number of neurons in the hidden layer needs to be adjusted according to experience and experimental feedback. Usually, the training starts from a small number (such as 25 to 50) and is adjusted according to the training results. If it is under fitted, gradually add more neurons; otherwise, if it is over fitted, consider reducing the number of neurons. As shown in Figure 3.3, this is a GRU model with three hidden layers, and the neurons in each layer are consistent. It can be seen that when training for the same data set, the different number of single-layer neurons (50, 100 and 150) will affect the convergence speed of the model. When the number of neurons increased from 50 to 100, we can see that the loss gradient decreased faster. However, there is a marginal effect of increasing neurons. When the number increases from 100 to 150, there is little difference in the gradient decline rate between the two. Therefore, the determination of the number of neurons needs to balance the calculation speed and accuracy.

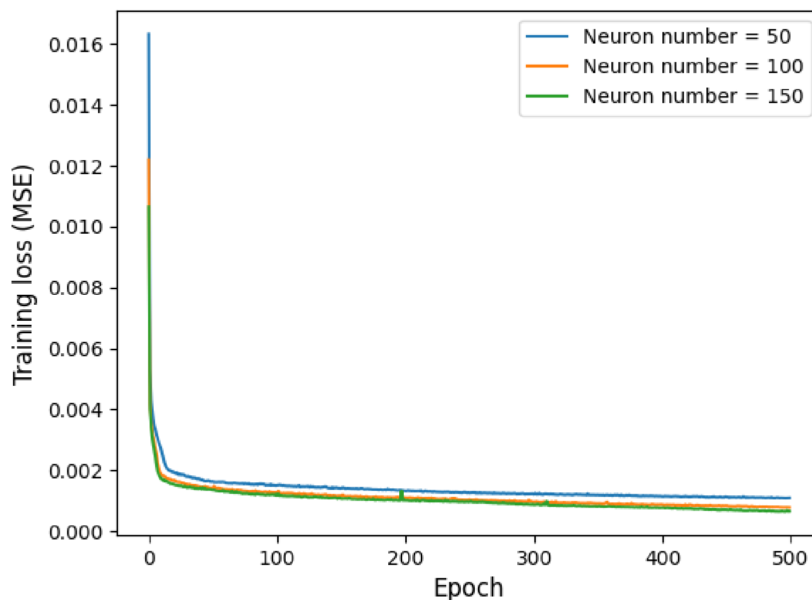


Figure 3.3: Comparison diagram of gradient descent of model

3.1.2.3 Selection of activation function

There are several aspects to consider when selecting an activation function. In terms of the representation ability of the network, the network can only represent the linear model without the activation function, and the ability is limited. Therefore, in order to make the network to have strong representation ability, the activation function needs to be nonlinear. Secondly, for forward calculation, we need to calculate the value of the activation function, and some functions are also functions about their own function value after derivation, so it is also very important to calculate simply. Moreover, in the process of back propagation, the gradient of the previous parameter update is generally the product of the derivative, which also includes the derivative of the activation function. If it is a function whose derivative value is related to the function value, such as Tanh or Sigmoid, there will be the activation function value in the derivative product. Therefore, we also hope to meet certain requirements for the value range of the activation function and the derivative of the activation function. It cannot be too large, otherwise it will lead to gradient explosion. It should not be too small, which will lead to gradient vanishing. Figure 3.4 shows the schematic diagram of various activation functions commonly used at present. The following will explain the three types of activation functions used in the project.

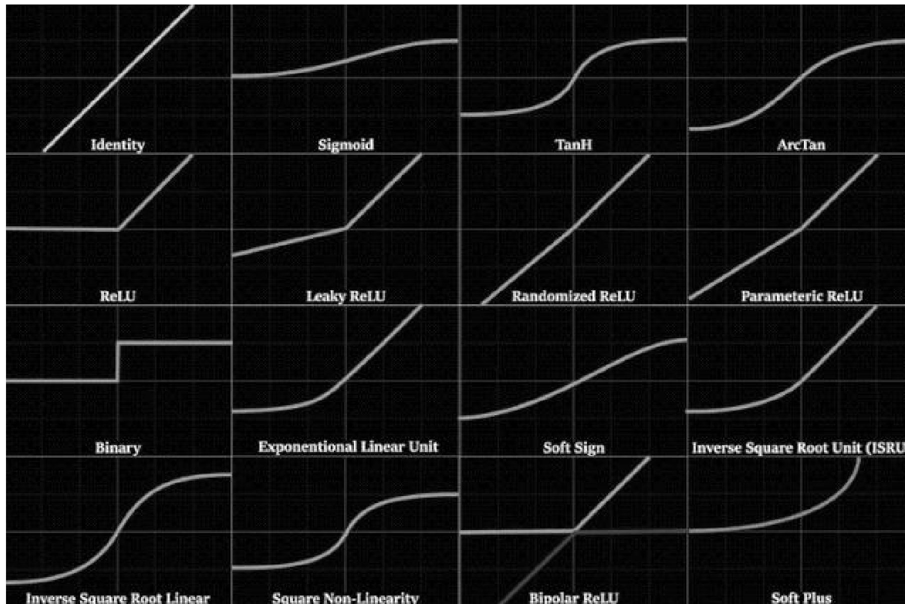


Figure 3.4: Schematic diagram of various activation functions

1. Sigmoid function
   Sigmoid function is a logistic function, as shown in Figure 3.5, which means that no matter what the input is, the output is between 0 and 1. In other words, each neuron and node input will be scaled to a value between 0 and 1. The function expression and derivative function are:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

$$Sigmoid'(x) = Sigmoid(x) \cdot (1 - Sigmoid(x)) \tag{3.3}$$

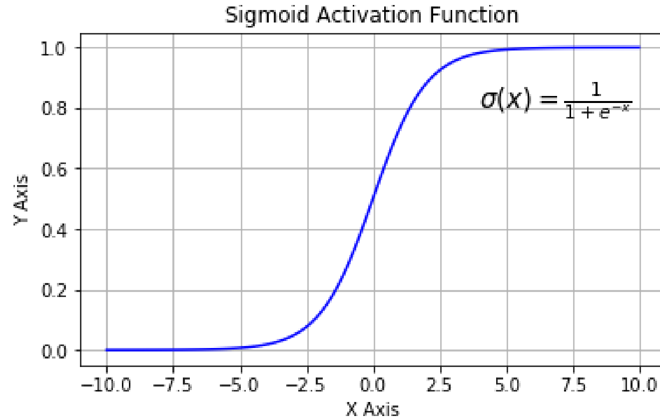


Figure 3.5: Sigmoid function graph

But Sigmoid has the following disadvantages:
1.The most obvious is saturation. It is not difficult to see from the above figure that the derivatives on both sides gradually approach 0, which is easy to cause the gradient to disappear.
2.Offset of activation function. The output value of Sigmoid function is greater than 0, so that the output is not the mean value of 0, which will cause the neurons in the later layer to get the non-zero mean signal of the previous layer as the input, which will affect the gradient.
3.The computational complexity is high because the Sigmoid function is exponential.

Because the output of Sigmoid is between 0 and 1, which conforms to the physical definition of gating, and when the input is large or small, its output will be very close to 1 or 0, so as to ensure that the door is opened or closed [13]. Therefore, the Sigmoid function is used in the memory selection unit of LSTM/GRU and attention mechanism of transformer as the activation function of gating.

2. Tanh function

Tanh activation function is also called hyperbolic tangent activation function, which is similar to the Sigmoid function. Tanh function also uses true value, but tanh function compresses it to the interval of - 1 to 1. Unlike Sigmoid, the output of tanh function is zero centered as shown in Figure 3.6, because the interval is between - 1 and 1. Think of the tanh function as two Sigmoid functions put together. In practice, the use of tanh function takes precedence over Sigmoid function [14]. When generating candidate state, LSTM and GRU models use tanh function because their output is consistent with the feature distribution with 0 center in most scenes. In addition, the tanh function has a larger gradient when the input is close to 0 than the Sigmoid function, which usually makes the model converge faster. Tanh function expression and derivative function are respectively,

$$tanh(x) = 2Sigmoid(2x) - 1, -\infty < x < +\infty, -1 < tanh(x) < 1 \tag{3.4}$$

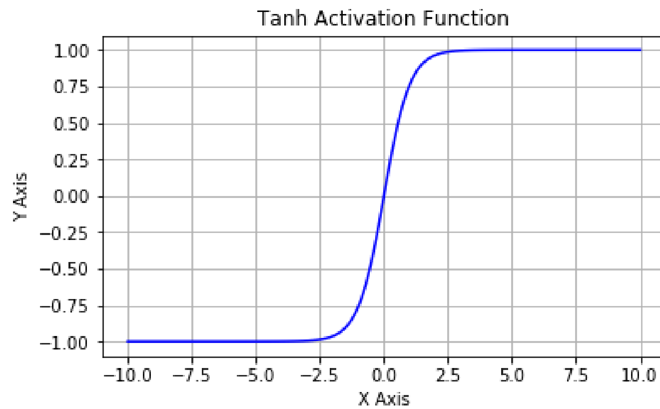


Figure 3.6: Tanh function graph

$$tanh'(x) = \frac{4}{(e^x + e^{-x})^2} \tag{3.5}$$

3. ReLU function
   ReLU function is actually a piecewise linear function. As shown in Figure 3.7, all negative values are changed to 0, while the positive values remain unchanged. It has several advantages [15]:
   1.ReLU has sparsity, which can make the sparse model better mine relevant features and fit the training data.
   2.In the region of $x > 0$, there will be no problem of gradient saturation and gradient vanishing.
   3.Low computational complexity, no exponential operation is required, and the activation value can be obtained as long as a threshold. Therefore, it is used in the full-connective layer connecting the LSTM/GRU layer and the output layer.

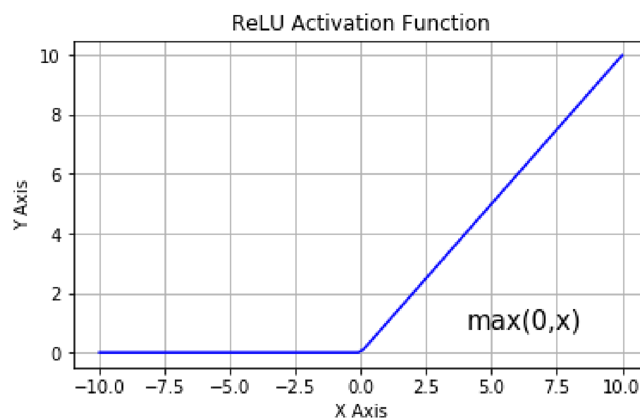


Figure 3.7: ReLU function graph

3.1.3 Determine the learning rate of neural network

It can be seen from 2.4 that in the training stage of neural network, adjusting the learning rate of gradient descent algorithm can change the update range of network weight parameters. When large loss and steep gradient are combined with learning rate, the next step will be large; When the loss is small and the gradient is relatively flat, combined with the learning rate, the next step will be shortened.

In order to make the gradient descent method to have better performance, we need to set the value of the learning rate in an appropriate range, because the learning rate determines whether the parameter can move to the optimal value and the speed at which the parameter moves to the optimal value [16]. As shown in Figure 3.8, if the learning rate is too large, the weight parameter is likely to exceed the optimal value, and finally jump back and forth on the side with the smallest loss, never stopping. Assuming that the learning rate is 1 and the optimal value is 0.5, the final loss value may jump back and forth between -0.5 and 1.5; On the other hand, if the learning rate is too small (for example, take 10e-5), the network may take a long time to optimize, and the optimization efficiency is too low, resulting in the inability of the algorithm to converge. Therefore, the learning rate is very important for the performance of the model.

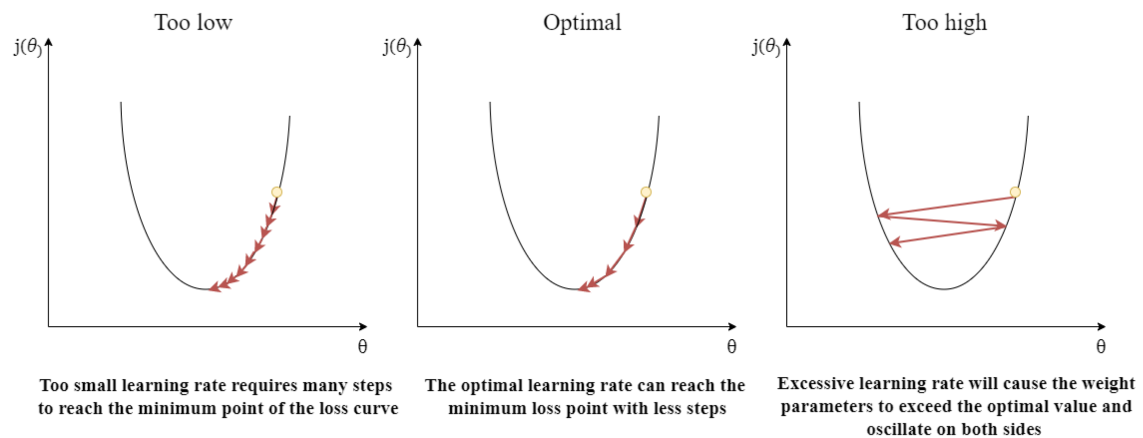


Figure 3.8: Influence of learning rate on loss gradient curve

The adjustment of learning rate can be summarized into the following steps, but in practice, it is also necessary to flexibly adjust the strategy in combination with experience:

1. First, we can choose an appropriate learning rate(like 0.01), and then try different learning rates for many times to check the downward trend of the network loss gradient at the beginning. We don't need to be too precise to determine the magnitude.
2. If the loss begins to decrease in the first few rounds of training, the magnitude of learning rate can be gradually increased until a value is found, so that the loss begins to fluctuate or increase in the first few rounds; On the other hand, if the loss curve fluctuates or increases at the beginning, try to reduce the magnitude until you find the setting that the loss decreases at the beginning of the round. Take half of the threshold to determine the learning rate.

3. Finally, in the fine tuning stage, adjust the learning rate of the network again in order to obtain the best calculation speed and accuracy.

3.1.4 Drop out and gradient clipping

The loss gradient is used to obtain the direction and amplitude of network parameter update in network training, and then update the network parameters with appropriate amplitude in the correct direction. In the deep network or RNN, the loss gradient accumulates in the update to obtain a very large gradient, which will greatly update the network parameters, resulting in the instability of the network. In extreme cases, the value of the weight becomes so large that the result will overflow (Nan value, infinite). In RNN, gradient explosion will lead to the instability of the network, which makes the network unable to learn well from the training data. Specifically, as shown in Figure 3.9, the loss gradient curve fluctuates seriously and cannot converge.

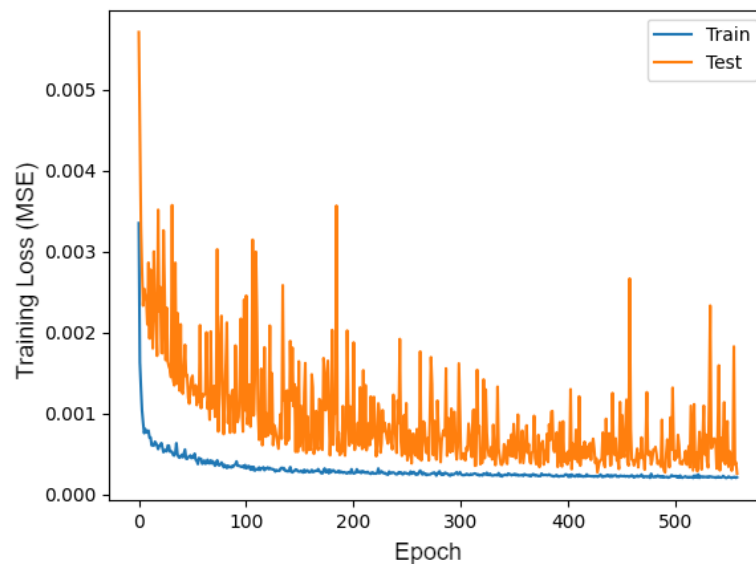


Figure 3.9: Neural network loss gradient explosion curve

Dropout is a common method to reduce the risk of over fitting in deep learning. When the neural network is trained, some neurons are inactivated like shown in Figure 3.10, which blocks the synergy between some neurons, thus forcing a neuron and randomly selected neurons to work together, reducing the joint adaptability between some neurons.

Dropout layer is adjusted by setting the dropout rate, which is the proportion of neuron inactivation in one hidden layer. In deep networks, using a dropout rate of 0.5 is a good choice, because a dropout layer can get the maximum regularization effect at this time; In shallow networks, the dropout rate should be lower than 0.2, because too much dropout rate will lead to the dropout of too much input data, which has a great impact on the model. Because three hidden layers network is used in the project, the best dropout rate is determined to be 0.15 after repeated training and comparison.

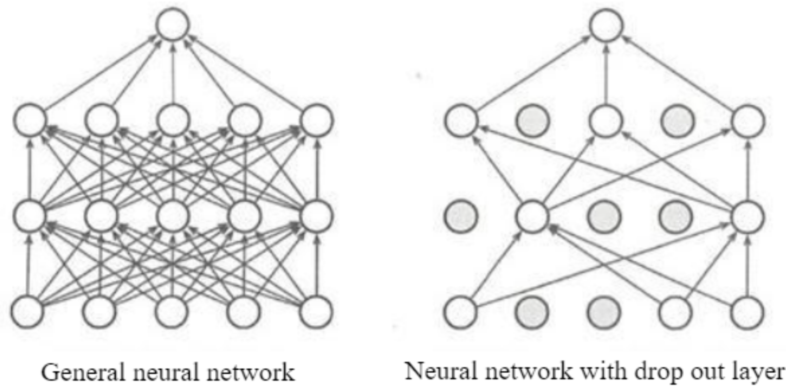


Figure 3.10: Comparison between standard neural network and neural network with dropout

Another method is also used to suppress gradient explosion. It can check and limit the size of gradient during network training, which is called gradient clipping. There are two common gradient clipping methods:

1. Determine a range. If the gradient of the parameter exceeds, cut it directly.
2. Cut when the L2 norm of the gradient exceeds the specified value.

3.1.5 Training epoch number and early stop mechanism

An epoch means that all data is sent into the network and completes a process of forward calculation and back propagation [17]. Generally speaking, an epoch is the process of training all training samples once. With the increase of the number of epochs, the number of weight update iterations increases, and the curve enters the optimal fitting state from the initial non fitting state, and finally enters the over fitting state.

How many epochs are appropriate? There is no absolute standard answer to this question. For different models, the convergence speed of the model is different. Even for the same model and parameter, with different data sets, the convergence speed and index value are quite different. How to adjust epoch needs to comprehensively look at the loss decline of the test set and the accuracy index of the validation set. The determination of epoch number can be divided into the following categories, but in practice, it needs to be adjusted in combination with the feedback results of the model:

1. If the loss of the training set can be reduced and the accuracy index of the validation set is still rising, it indicates that it is under fitting and has not converged. We should continue to train and increase the epoch number.
2. If the training set loss can be reduced again, and the accuracy index of the validation set is also decreasing, indicating that's over fitting, we must take measures to prevent over fitting, such as early stopping or reducing the number of epochs.
3. If the preset number of epochs is 100, by the 50th epoch, the loss of the training set will not decrease, and the indicators of the validation set will not move, indicating that 50 epochs are enough, and the accuracy of the model

has stagnated. (However, there is another possibility in this case: the model has only reached the local optimum, and there is no global optimum. It may try a larger batch size, and the loss gradient will continue to decline in one direction, and the accuracy index can go up again.)

Early stop is a mechanism to prevent the neural network from over fitting the data during training. When we see that the training and validation loss map begins to diverge, we terminate the training through this mechanism. This is usually done in the following two cases:

1. In the process of training, the training model can be used to predict the validation set every certain amount of step. After a certain number of training steps, the error of the model in the training set is still decreasing, but the error in the validation set is not decreasing. At this time, the model is over fitted, and the rising threshold of the error gradient of the validation set is set as the trigger condition of the early stop mechanism.
2. In another case, as shown in Figure 3.11, when reducing the learning rate and increasing training are not helpful to the model, the loss gradient of both the training set and the validation set remains unchanged after a certain number of epochs (the red box in the figure), which requires the early stop mechanism to stop training in time.

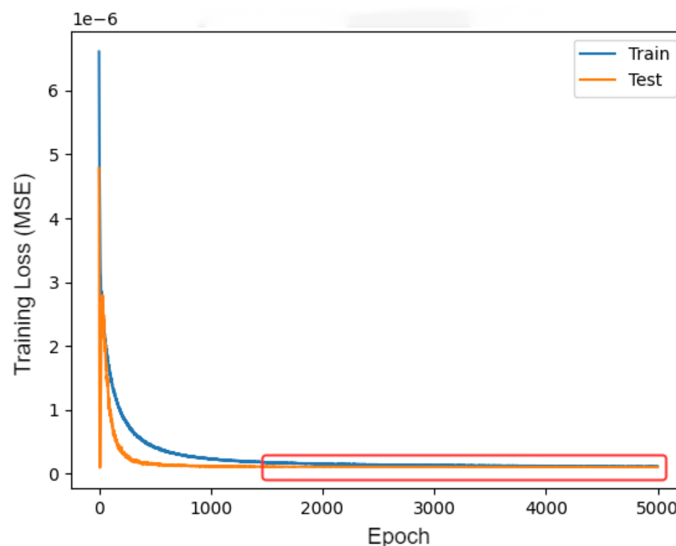


Figure 3.11: Model loss gradient descent stagnation diagram

3.2 Problems encountered in model training and Countermeasures

3.2.1 The influence of batch size on the prediction accuracy of the model

3.2.1.1 Reasons for adopting large batch size

Because the training data is often too large and limited by the performance of computer hardware, it is impossible to send all the data to the network for calculation at one time. Therefore, the training data is divided into several parts, that is, it is divided into multiple batches and sent to the network computing one by one. The batch size is the number of data samples handled by the neural network in one training period. For example, suppose the training set has 1000 samples. When the batch size is 10, 100 times iteration of the model are required to train a complete sample set (i.e. finished 1 epoch). Using large batch size has the following advantages [18]:

1. The memory utilization is improved, and the parallelization efficiency of large matrix multiplication is improved.
2. The number of iterations required to complete an epoch (full data set) is reduced, and the processing speed of the same amount of data is further accelerated.
3. Within a certain range, generally, the larger the batch size is, the more accurate the descending direction is, and the smaller the training oscillation is caused like shown in Figure 3.12.

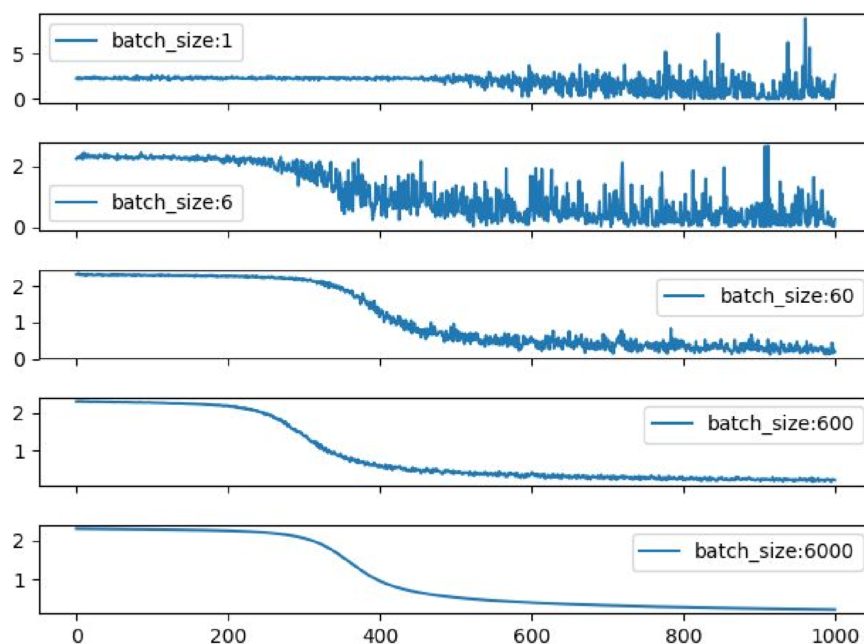


Figure 3.12: Smoothness of loss gradient under different batch sizes

In the early stage of the project, due to the limitation of hardware (mainly reflected in the insufficient graphics memory of the graphics card), only a small batch size (batch size = 256) can be used for training. With the strengthening of hardware conditions in the medium stage, the hardware platform can provide stronger computing performance and larger graphics memory space, and it is possible to increase the batch size. After repeated attempts, the batch size can be increased to 4096.

The biggest improvement after using a large batch size lies in the great shortening of training time. When using a small batch size(batch size = 256), the duration of a single training is usually two to three hours, while when using the same model parameters and training data after increasing bacth size(batch size = 4096), the training time is shortened to 15 minutes.

3.2.1.2 Problems encountered in using large batch size

The shortening of training duration greatly increases the training efficiency and accelerates the project progress, but there are other problems after increasing the batch size.

The number of iterations required for the model to complete an epoch with a large batch size is reduced, but this makes the time spent to achieve the same accuracy increase, so the correction of parameters becomes more slow; Another potential problem is that when the batch size increases to a certain extent, the determined decline direction basically does not change, which will affect the introduction of randomness and weaken the generalization ability of the model. Figure 3.13 and Table 3.3 show the performance of the same model parameters and training set under different batch sizes.

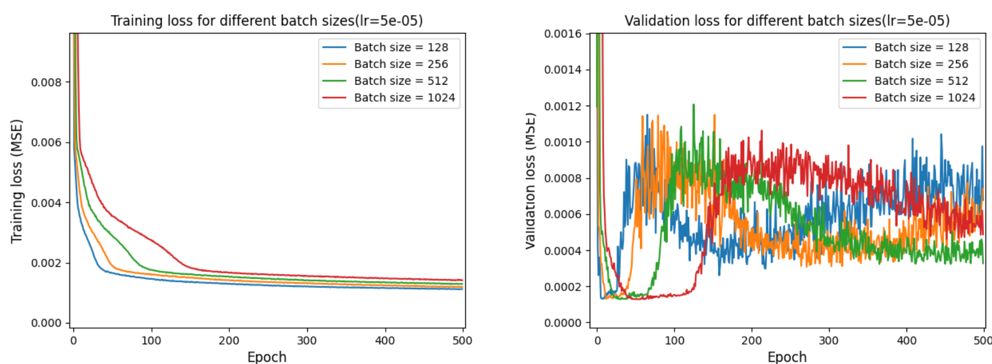


Figure 3.13: Loss curves of different batch sizes

Table 3.3: Minimum values reached by different batch sizes

Batch size	Minimum training loss	Minimum validation loss
128	0.026394	0.001898
256	0.04402	0.006919
512	0.073988	0.007238
1024	0.095214	0.014617

From the above figure, we can conclude that the larger the batch size is:

1. The slower the training loss decreases as shown by the slope difference between the red line (batch size = 1024) and the blue line (batch size = 128).
2. The higher the minimum validation loss. For example, the minimum validation loss is 0.014617 for a batch size of 1024 and 0.001898 for a batch size of 128.
3. The less time it takes to train in each iteration but more epoch is required to converge to the minimum validation loss. When using a large batch size for training, each epoch takes less time. When the batch size is 128, each epoch takes 3 seconds, while when the batch size is 1024, it takes 0.5 seconds, which reflects the lower overhead associated with loading a small number of large batches. However, large batch size requires more epoch to converge to the minimum value. Therefore, it may take longer on the whole.

3.2.1.3 Solution of large batch size problems

Keskar et al proposes an explanation [19] for the problems described in the previous section: training in large batch size is easy to converge to sharp minimization, and the gradient curve of the minimization is very steep, so that the movement of the minimization in a small range will greatly change the loss gradient, while a small batch size tend to converge to a flat minimization, and the change of the loss gradient is smoother. Flat minimizers tend to be more generalized because they are more robust to changes between training and test sets.

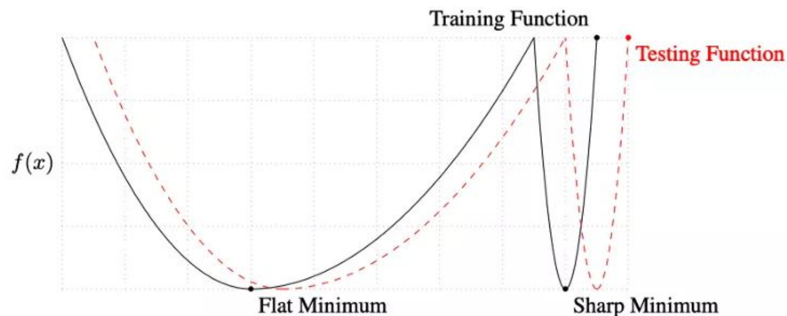


Figure 3.14: Concept diagram of flat and sharp minimum

Goyal et al proposed a solution: "Linear scaling rule" [20], that is, when the batch size is expanded by K times, the learning rate used is also multiplied by K. The article concludes that if a learning rate under a certain batch size can make the model trained to a stable accuracy, the change of batch size and learning rate can also train the model to a similar accuracy as long as they follow the linear scaling rule.

According to this rule, under the same model parameters and training set, four batch sizes are still adopted, in which the learning rate of batch size 128 is 5e-05, and the learning rate of the other batch sizes is increased proportionally. Figure 3.15 and Table 3.4 shows the performance of the model under different batch sizes in this condition.

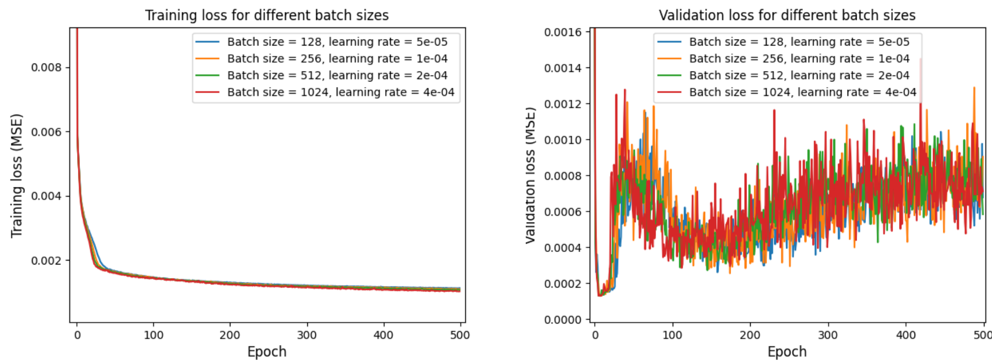


Figure 3.15: Loss curves of different batch sizes

Table 3.4: Minimum values reached by different batch sizes

Batch size	Minimum training loss	Minimum validation loss
128	0.026394	0.001898
256	0.028822	0.00194
512	0.028553	0.002026
1024	0.026754	0.002261

Through the data comparison between the above picture and table, it can be found that adjusting the learning rate does eliminate most of the performance gap between a small batch and a large batch. Now, the validation loss of batch size = 1024 is 0.002261 instead of 0.014617, which is closer to the loss of 0.001898 when the batch size is 128. It can also be seen from the loss curve that after adjusting the learning rate, no matter which size of the batch, the training loss and validation loss curve are close to coincidence.

Therefore, this method is used for subsequent training, so that the training speed and parallelization ability of the model can be improved by using large batch size, while taking into account the convergence speed and error accuracy of the model.

3.2.2 The problem of output weight assignment under multiple output task

It can be seen from section 1.3.1 that the neural network model needs to deal with a multi input and multi output task. However, when dealing with such multi task learning, there is a serious problem, that is, the scale difference between the outputs is huge, and the response in model training is that, as shown in Table 3.5, the reduction speed of the loss function of each output is very inconsistent. Often, one item decreases very fast and the other item decreases super slowly. In the process of loss convergence leading to different outputs, the gradient size is different, and the sensitivity to different learning rates is also different. The loss with small gradient is taken away by the loss with a large gradient, which weakens the generalization ability of the learned features.

Table 3.5: Hyperparametric sensitivity of different outputs

Output type	Learning rate	Validation loss	Converge epoch
Power consumption	0.01	5e-04	300
Compressor inlet temperature	5e-04	7e-07	200
Evaporator mean temperature	1e-04	5e-08	160
WCC inlet pressure	1e-05	1e-09	300

The first solution to this problem is to adopt different learning rate and loss function weight distribution for different outputs, in order to unify each loss to an order of magnitude as far as possible.

Assuming that the final loss function of the neural network model has two contributions: $l(\theta) = f(\theta) + g(\theta)$, these two terms, $f(\theta)$ and $g(\theta)$, correspond to the outputs of two different metrics respectively. With the progress of learning, the reduction speed of the two loss functions is very inconsistent. Use different learning rates for different loss items, e.g. adaptive learning rate. For example, in the Adagrad algorithm, the gradient descent algorithm formula for adaptive learning rate is:

$$\theta = \theta - \frac{lr}{\sqrt{G_{tt} + \varepsilon}} \frac{\partial l}{\partial \theta} \tag{3.6}$$

where lr is the learning rate and $\frac{\partial l}{\partial \theta}$ is the gradient of the loss function to the network parameters. Here, the learning rate is divided by an adaptive constant $\sqrt{Gtt + \varepsilon}$, where $\varepsilon(\varepsilon = 1.8)$ is a small constant value to prevent zero division divergence. Gtt is a diagonal matrix, and its ith matrix unit corresponds to the square of the gradient $\frac{\partial l}{\partial \theta_i}$ along the ith direction, which is the superposition of time 0 to t.

For multi task learning, the above adaptive learning rate is not useful. If expanded according to $\frac{\partial l}{\partial \theta} = \frac{\partial f}{\partial \theta} + \frac{\partial g}{\partial \theta}$, the adaptive learning rate is used in multi task learning in the following form:

$$\theta = \theta - \frac{lr}{\sqrt{G_{tt}+\varepsilon}} \left(\frac{\partial f}{\partial \theta} + \frac{\partial g}{\partial \theta}\right)$$

If the magnitude of $\frac{\partial f}{\partial \theta}$ and $\frac{\partial g}{\partial \theta}$ is inconsistent, the above equation cannot provide adaptive learning rate for different loss function terms. According to this idea, we can require a smaller learning rate for tasks with fast convergence and a larger learning rate for tasks with slow convergence:

$$\theta = \theta - \frac{lr}{\sqrt{F_{tt}+\varepsilon}} \frac{\partial f}{\partial \theta} - \frac{lr}{\sqrt{G_{tt}+\varepsilon}} \frac{\partial g}{\partial \theta}$$

where Ftt and Gtt are the time accumulation of the gradient square of different output items respectively.

The method of adaptive learning rate has achieved good results when there are only few output items at the beginning. However, when there are 8 training output items in the later stage, there is a situation similar to zero sum game in the model: the income of one side will inevitably bring the loss of the other side. When the model assigns more weight to one input, the training effects of almost all other outputs are negatively affected, So that the loss weight of the whole model will worsen no matter how it is allocated.

In order to completely solve this problem, another method is adopted as the final solution: neural network modularization, different tasks are completed in different neural network modules, and there is no interference between the resulting tasks.

As shown in the Figure 3.16, a neural network can be decomposed into many sub networks, and different network combinations are used in training and deployment. Each sub network predicts only one output index according to the input, and then integrates all sub networks through a packaging program.

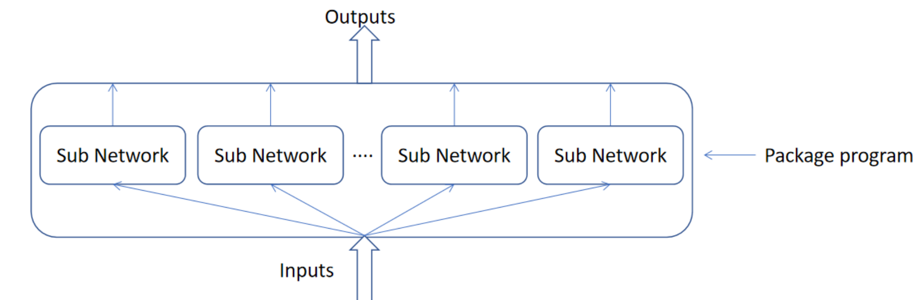


Figure 3.16: Schematic diagram of modular neural network

In this way, each sub network can be tuned for a specific output to make the output accuracy as good as possible. At the same time, flexibility is taken into account, when new output needs to be added, only the sub network for this output needs to be trained and added to the package program. So the operation of the original functions is not affected, and new functions can be added independently, without considering the loss weight distribution between different outputs.

3.3 Determine the type of neural network used for the project

At present, there are more than ten mainstream neural network models widely used in various fields. If we consider the variant models developed based on these basic models, the number may exceed hundreds. The mathematical principles and architectures of different neural network models are quite different, so these models have their own fields of expertise. For example, FCN is mainly used in the classification task of digital recognition, RNN is widely used in the fields of speech analysis, character analysis and time series analysis, and CNN is mainly used in graphic analysis and processing. Therefore, when neural network is used to predict the performance of vehicle thermal system, it is necessary to determine which specific neural network model is used for further development.

The most direct method is to get the performance gap between different models by comparison. All models are trained based on the same data set, using the same model configuration (the same number of hidden layers and neurons) and tuning strategy. After training, predict the same test set, the performance gap of the model is judged through the prediction accuracy of each model, and the model with the best performance for further development is selected.

Considering that the power consumption of a vehicle thermal system is a very important index, which directly affects the driving range of the vehicle, this index is used as the prediction object of model comparison, that is, the smaller the error between the predicted value of power consumption and the experimental value, the

better the performance of the model. The input of the neural network model takes the performance parameters of each vehicle component. The specific input and output elements of the model are shown in Table 3.6.

Table 3.6: Dataset input and output elements for model comparison

	Parameters	Parameters refer to components	I/O Classification
#1	Pump speed	Compressor, condenser, evaporator, radiator,	Input
#2	Inlet temperature	ED pump , battery pump	
#3	Inlet pressure	Compressor, condenser, evaporator, radiator	Input
#4	Power consumption	Total power consumption of thermal system	Output

After selection, six kinds of neural network models are selected for training and comparison: FCN, CNN, RNN, LSTM, bi-LSTM and GRU. All models use two layers of hidden layers, in which the first layer contains 100 neurons and the second layer contains 80 neurons. In the training process, all models have been tuned for many times and optimized as much as possible to have the best effect. See section 3.1 for specific tuning strategies. The final prediction results of each model are shown in Figure 3.17. (Reminder: due to confidentiality requirements, the displayed data has been normalized and is not real experimental data.)

At the same time, the root mean square error(RMSE) is used to calculate the error of the predicted value and experimental value of each model. The smaller the RMSE value calculated from

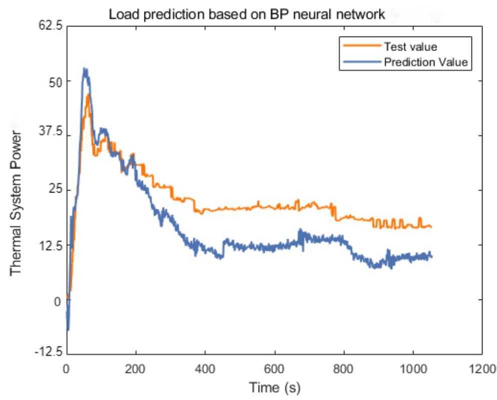
$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2} \tag{3.7}$$

the higher the accuracy.

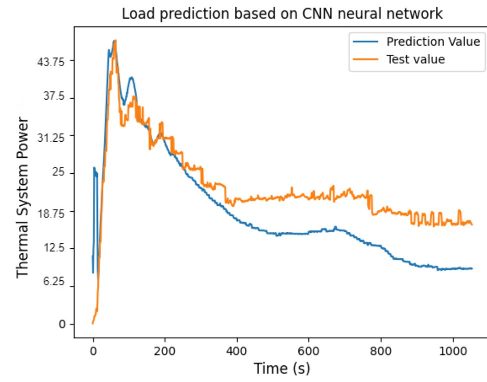
Table 3.7: RMSE value of each model

Model type	GRU	biLSTM	LSTM	RNN	CNN	FCN
RMSE	**203.3**	248.5	264.1	389.5	474.5	1562.9

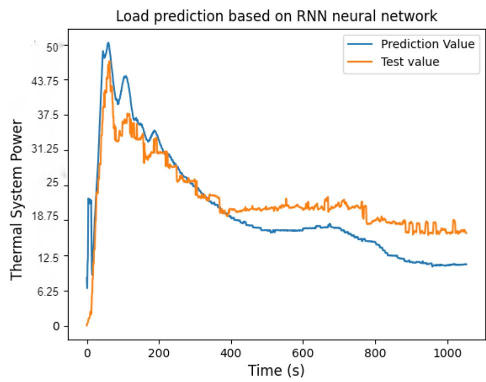
It can be seen from the Table 3.7 that the RNN family has good performance in time series prediction projects, and the GRU model, as a variant of RNN, has the best prediction accuracy in result comparison. Therefore, the GRU model is adopted as the main model of the project for further development. The performance of CNN and FCN models is far from that of RNN models. Especially, as the most primitive neural network, the prediction error of FCN is much greater than that of other models.



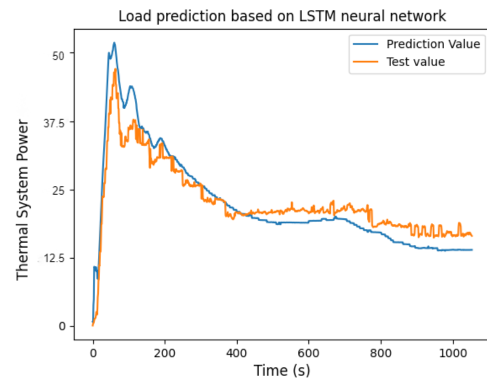
(a) FCN model prediction curve



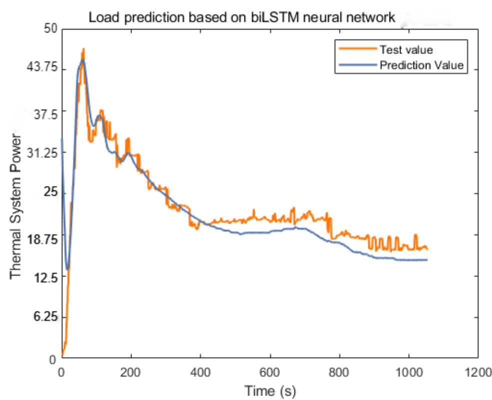
(b) CNN model prediction curve



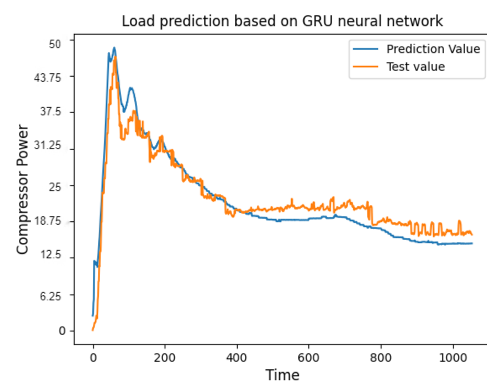
(c) RNN model prediction curve



(d) LSTM model prediction curve



(e) bi-LSTM model prediction curve



(f) GRU model prediction curve

Figure 3.17: Prediction curve of each model

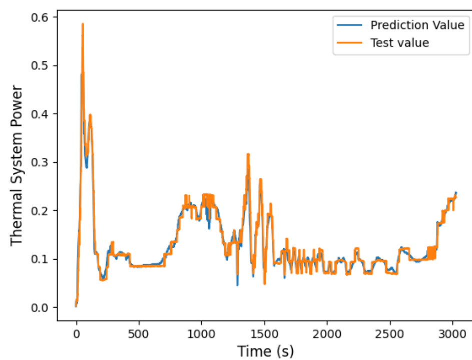
4

Neural network prediction results and comparison

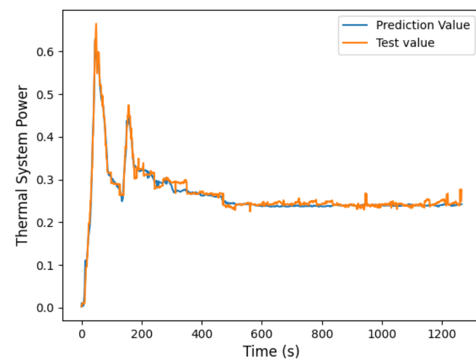
This chapter will show and discuss the final prediction results of the neural network model after training, and compare them with the experimental values. The final result is predicted by the GRU model with self-attention mechanism, hereinafter referred to as the GRU model. The display results come from the prediction of GRU model made on a part of the test set. The test set does not participate in the training of the model, so its information is unknown to the model, which can effectively test the accuracy and generalization of the model's prediction of strange information. The test set covering various working conditions, such as the ambient temperature from -10°C to 30°C, including soaked and steady road conditions. In addition, due to the confidentiality requirements of experimental data, all experimental values and predicted values are normalized, so the original data is not shown directly.

4.1 Prediction of power consumption of vehicle thermal system

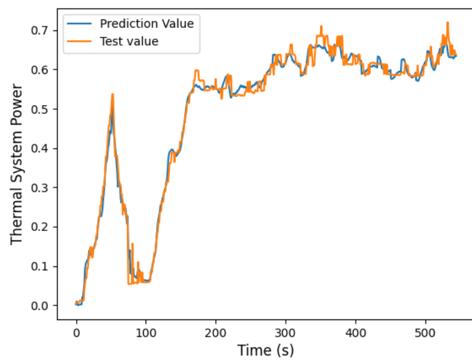
It can be seen from Figure 4.1 that no matter which data set, GRU model shows high prediction accuracy for power consumption, and even in most cases, the prediction curve coincides with the actual curve. As the most important performance index of the thermal system, the high-precision prediction of the model is helpful to truly reflect the actual situation of vehicle thermal system.



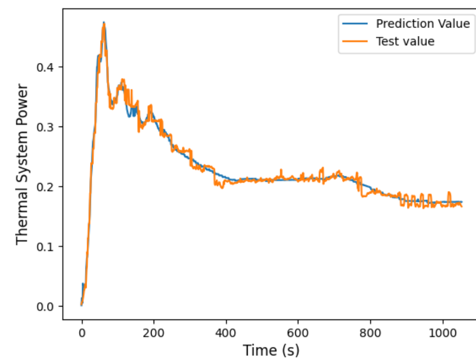
(a) Test set 1



(b) Test set 2



(c) Test set 3

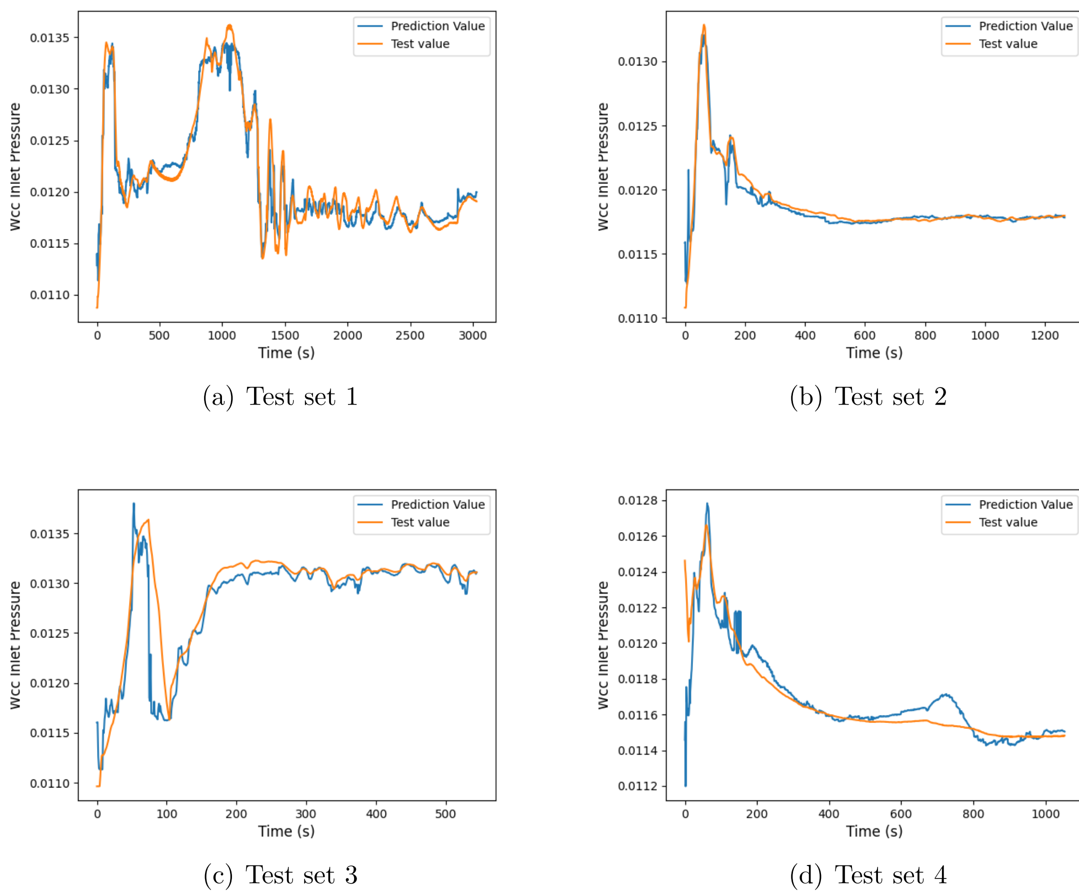


(d) Test set 4

Figure 4.1: Prediction results of GRU model for each test set

4.2 Prediction of inlet pressure of water cooling condenser

As can be seen from Figure 4.2, the GRU model still maintains very high accuracy in predicting the inlet pressure value of WCC. However, in test set 4, the prediction curve is quite different from the experimental value curve in the early stage. Combined with the analysis of the power consumption prediction curve of this test set, it is considered that the self-attention mechanism may project too many weights to the power consumption, resulting in the maximum input weight of the power consumption, which will affect the predicted value.



(a) Test set 1

(b) Test set 2

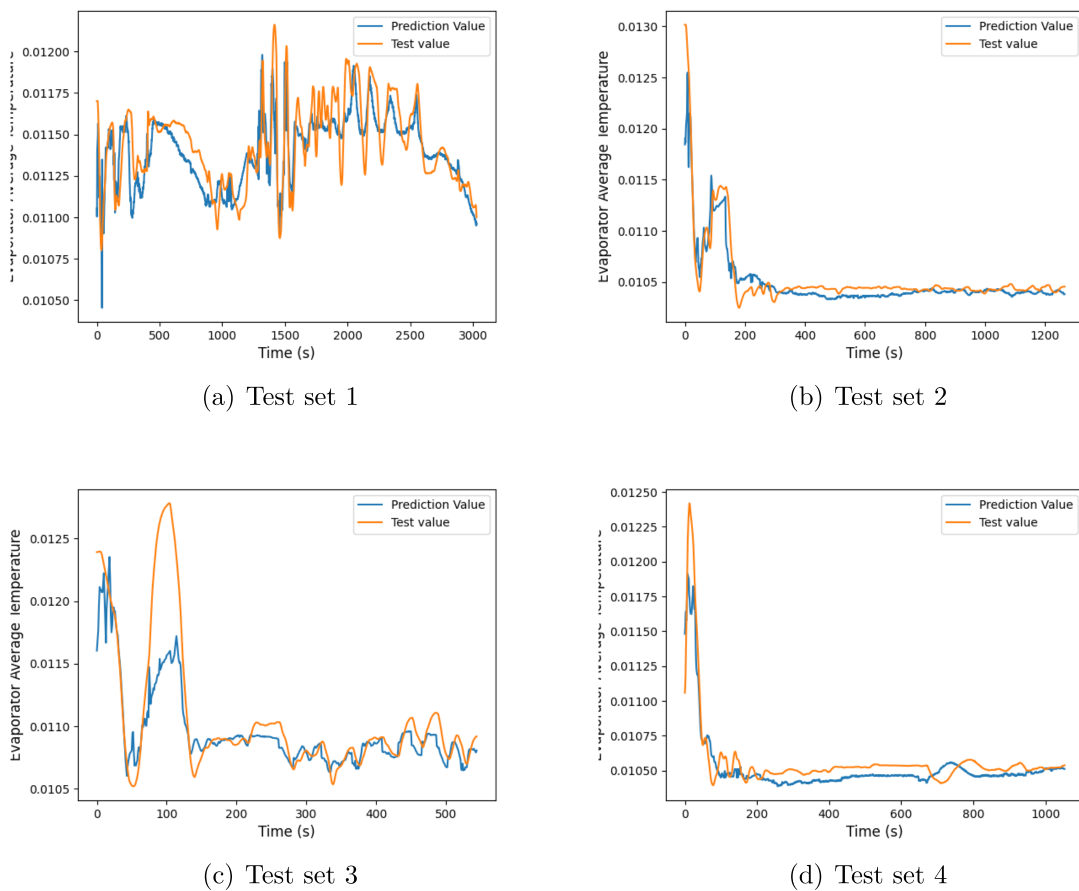
(c) Test set 3

(d) Test set 4

Figure 4.2: Prediction results of GRU model for each test set

4.3 Prediction of average temperature of evaporator

The prediction of evaporator average temperature value is shown in Figure 4.3. The GRU model still maintains high accuracy in the prediction of this parameter, but it can be seen from the observation of test set 3 that the actual value of this test set has a sharp rising peak, but the GRU model can not predict this peak, and then this is the best result after repeated tuning. It is speculated that this phenomenon stems from the regulation of evaporator valve in a very short time, and the model is difficult to predict this change



(a) Test set 1

(b) Test set 2

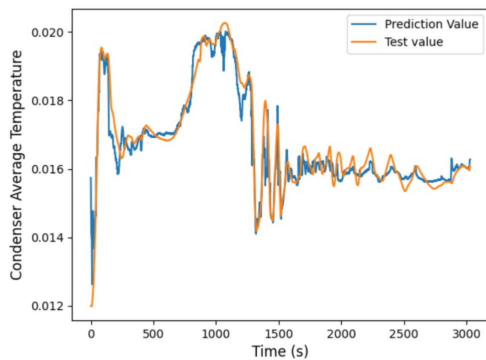
(c) Test set 3

(d) Test set 4

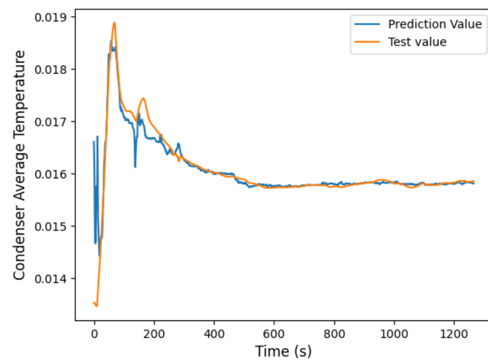
Figure 4.3: Prediction results of GRU model for each test set

4.4 Prediction of average temperature of condenser

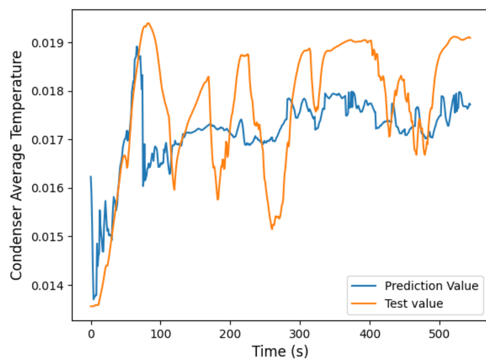
Figure 4.4 shows the prediction of average temperature of condenser by the GRU model. Test sets 1, 2 and 4 have good performance, while the predicted value of test set 3 has a large deviation from the actual value, and the predicted curve can not keep up with the fluctuation of the actual curve. However, the prediction curve can be regarded as an average of the actual curve, so it still has practical significance in this case.



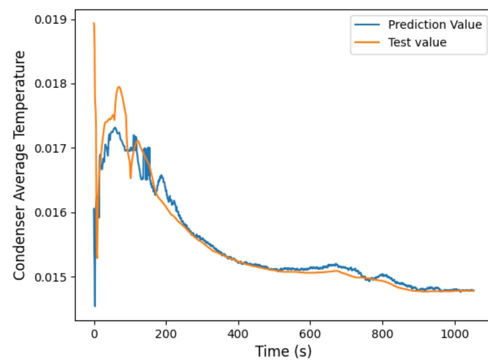
(a) Test set 1



(b) Test set 2



(c) Test set 3

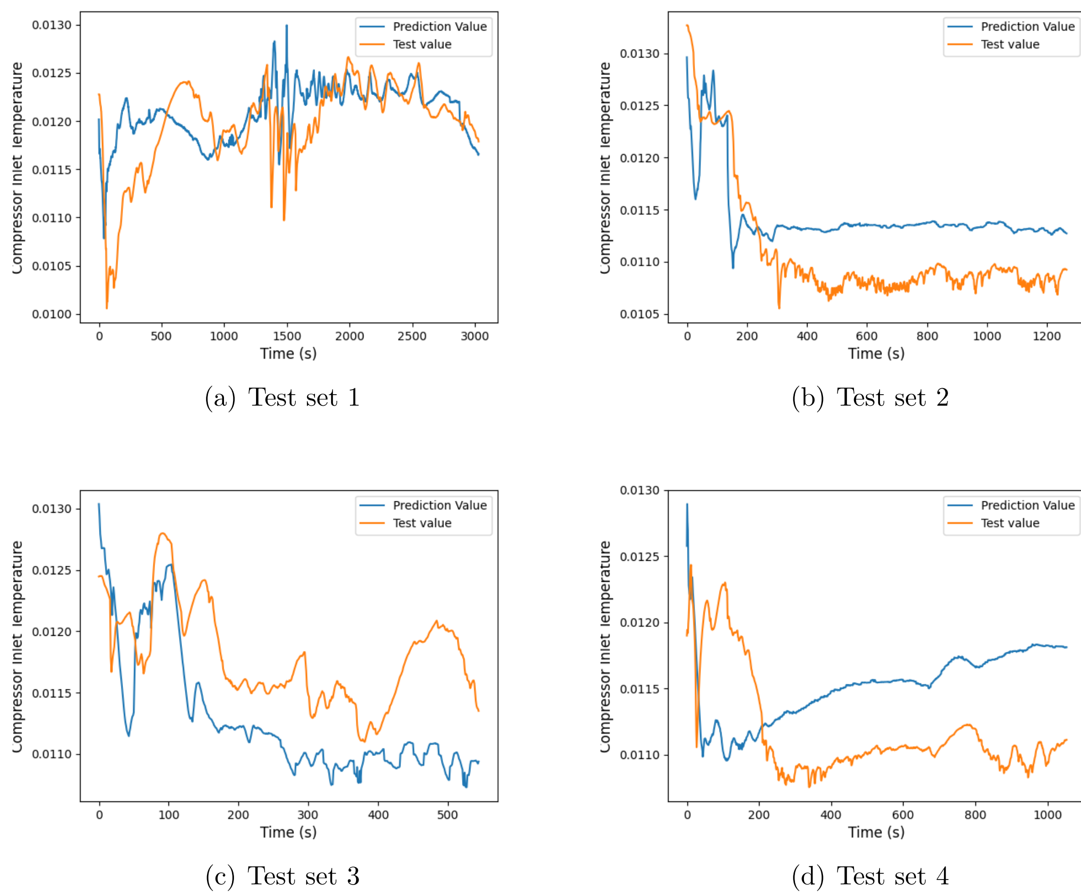


(d) Test set 4

Figure 4.4: Prediction results of GRU model for each test set

4.5 Prediction of compressor inlet temperature

It can be seen from Figure 4.5 that the GRU model predicts the inlet temperature of the compressor. It can be seen that the prediction accuracy of the GRU model for this parameter is not very good. The compressor generally enters the air through connection with the external environment, so there should be a certain relationship between the inlet temperature and the ambient temperature. However, after comparing the compressor inlet temperature and ambient temperature, it is found that there is a large difference between them, which is speculated to be caused by the vehicle structure. This may lead to the problem that the prediction accuracy of GRU model with ambient temperature as input is not high enough.



(a) Test set 1

(b) Test set 2

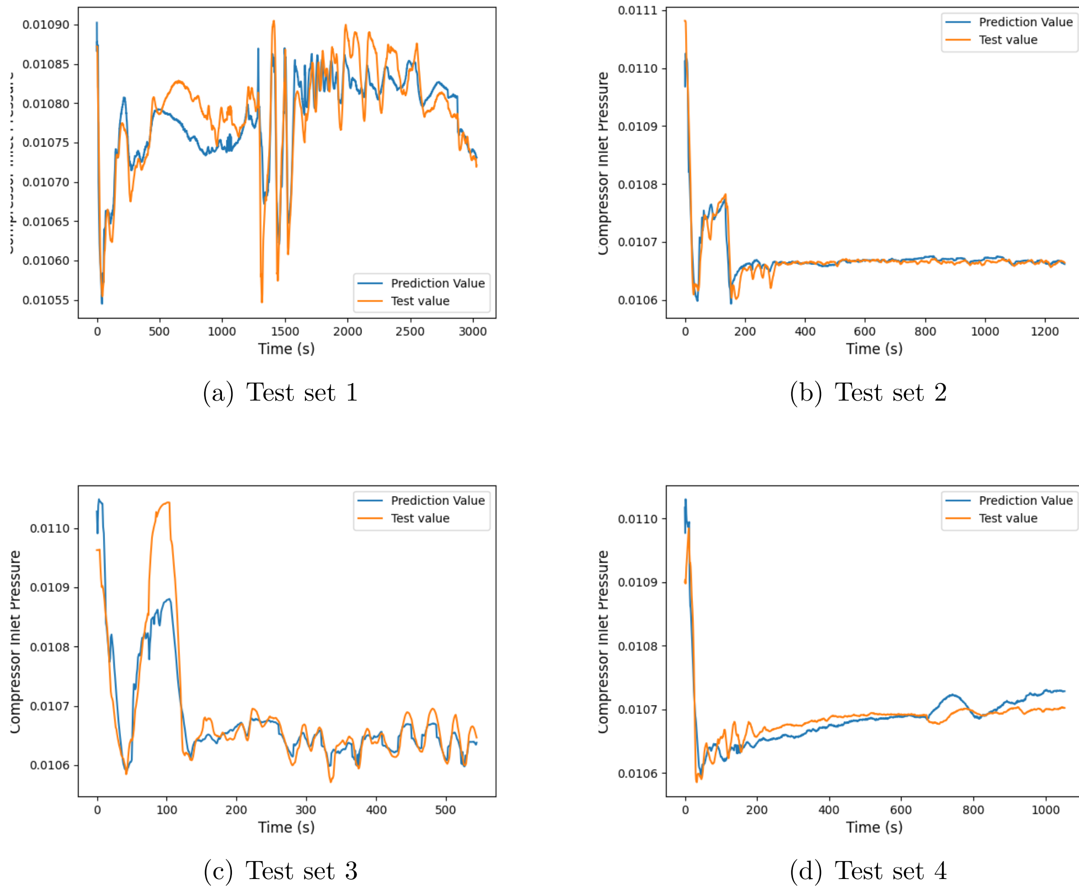
(c) Test set 3

(d) Test set 4

Figure 4.5: Prediction results of GRU model for each test set

4.6 Prediction of compressor inlet pressure

It can be seen from Figure 4.6 that the GRU model predicts the inlet pressure of the compressor. Differently from the inaccurate prediction of compressor inlet temperature, the prediction accuracy of the GRU model for inlet pressure is satisfactory. An interesting point is that the curve of this parameter is very similar to the curve of the average temperature of the evaporator(Figure 4.3), and there seems to be some connection between the two.



(a) Test set 1

(b) Test set 2

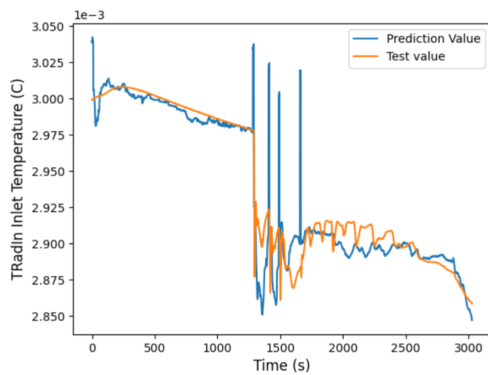
(c) Test set 3

(d) Test set 4

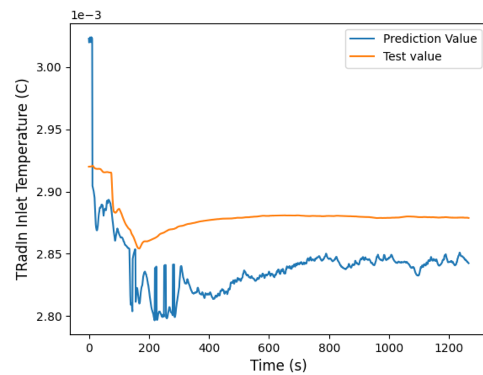
Figure 4.6: Prediction results of GRU model for each test set

4.7 Prediction of radiator inlet temperature

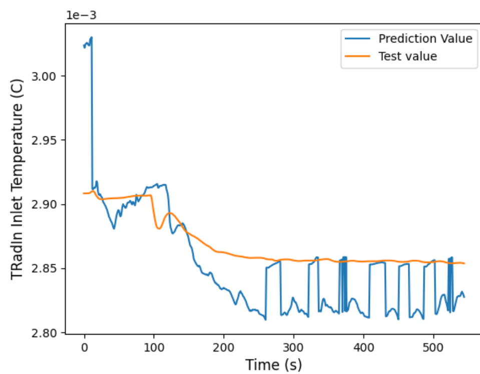
Figure 4.7 is a comparison diagram of the predicted inlet temperature of the radiator. Although the difference between the predicted curve and the actual curve is large from the image, the actual error is very low because the difference between the predicted value and the experimental value is very small.



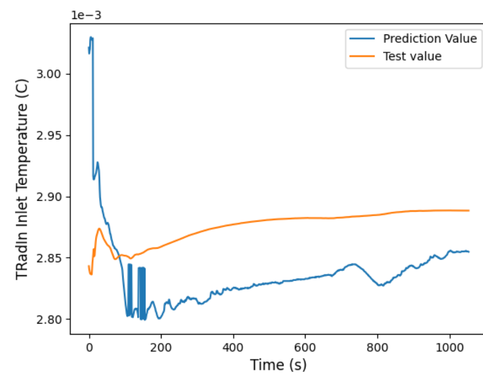
(a) Test set 1



(b) Test set 2



(c) Test set 3

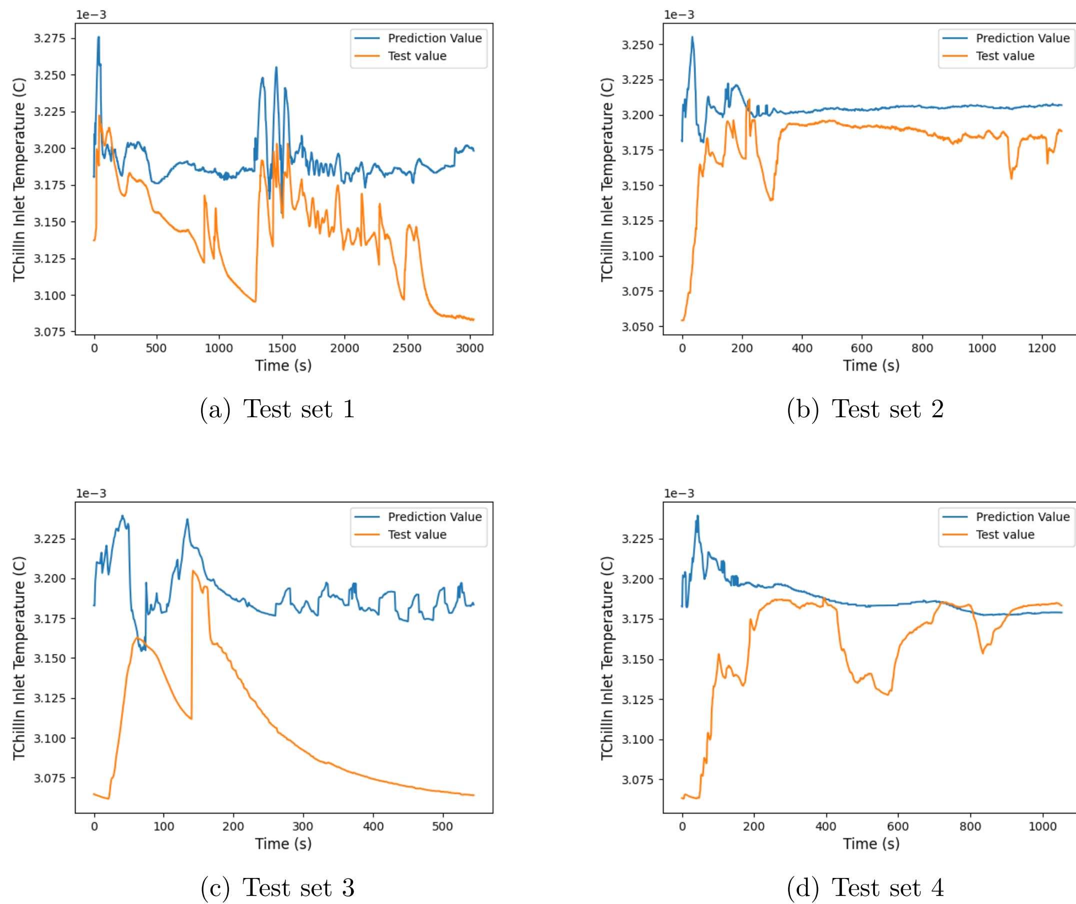


(d) Test set 4

Figure 4.7: Prediction results of GRU model for each test set

4.8 Prediction of chiller inlet temperature

Similar to the prediction of radiator inlet value, the prediction curve of chiller inlet temperature is quite different from the experimental curve from the image(Figure 4.8), but because the real values of the two are very close (the error is less than 1°C), the prediction accuracy of the model can be guaranteed.



(a) Test set 1 (b) Test set 2

(c) Test set 3 (d) Test set 4

Figure 4.8: Prediction results of GRU model for each test set

5 Discussion

5.1 Future work

The neural network model is trained based on the existing experimental data, which are tested based on the WLTC standard. Therefore, under other test standards, such as NEDC, FTP-75 and even random driving, the prediction accuracy and generalization performance of neural network still need to be verified. If we can add new experimental data next, especially the experimental data under different standards, it will be a good verification of the robustness of neural network.

At the same time, the original purpose of training a neural network model is to supplement the existing physical model. However, due to the complex structure of the existing physical model, it is difficult to handle, and it can not be used for unauthorized reasons, there is no comparison between the performance of neural network model and physical model. Therefore, we can not get the result whether the neural network is better than the physical model in prediction accuracy and robustness. So next, if we have the opportunity, we can run the neural network model and physical model at the same time under the same input conditions, and judge their performance through their output results.

Then, in the training process of a neural network, the comparison object has always been the experimental data, which means that the prediction error value of the model is only the difference between the predicted value and the experimental value at the current time, and the error at the previous time will not be considered. When the trained neural network model is used for closed-loop simulation, the controller will adjust the control strategy according to the predicted value given by the model, so as to affect the subsequent simulation. Therefore, there may be error accumulation in this process, and whether this will cause the simulation results to deviate significantly from the correct value over time remains to be verified later.

5.2 Ethical and sustainability impacts of the project

This neural network prediction model is used in the early development stage of automobile, and its main purpose is to speed up the simulation of automobile thermal system. Therefore, it is difficult to say that there is a direct ethical impact. However, after using this method, the simulation process of automotive CAE can be accelerated, and the difficulty and workload of testing and debugging can be reduced. Therefore, it may objectively reduce the workload and pressure of relevant personnel, which is its potential ethical impact.

From the perspective of sustainability, with the adoption of the neural network model, engineers can simulate the performance of the automotive thermal system under more working conditions, evaluate the performance of the thermal system more comprehensively, and optimize its structure and control strategy due to the reduction of test difficulty and the shortening of simulation time. This may make the performance of the thermal system better and more efficient, so as to improve the energy efficiency and service life of the vehicle.

6
Conclusion

In the process of automobile development, the cost of experimental manpower and time is huge, and most of the work is completed by computer simulation. The traditional physical model has the problems of cumbersome modeling steps and large difference between simulated value and real value. This thesis work attempts to introduce neural network model to predict the performance of vehicle thermal system in order to speed up the development process in early phases. And produce TEM loads for many cases, improve the capability.

Among various kinds of neural network models, the GRU model performs best because it has memory units, which can capture the long-range dependence between input and output. At the same time, the units with selective properties can screen important historical input information to improve the robustness of the model. In the model optimization stage, the self attention mechanism is also introduced for the GRU model, which can automatically identify the inputs with high correlation in multi-dimensional inputs and assign higher weights to improve the prediction accuracy of the model.

When the final model is tested with the data set of unknown data, it maintains good accuracy in the output of all dimensions, especially in the important prediction indicators represented by power consumption. At the same time, the test set includes various working conditions, which also proves the generalization and robustness of the model.

Bibliography

[1] W. Wu, S. Wang, W. Wu, K. Chen, S. Hong, and Y. Lai, "A critical review of battery thermal performance and liquid based battery thermal management," *Energy conversion and management*, vol. 182, pp. 262–281, 2019.

[2] D. Choi, Y. An, N. Lee, J. Park, and J. Lee, "Comparative study of physics-based modeling and neural network approach to predict cooling in vehicle integrated thermal management system," *Energies*, vol. 13, no. 20, p. 5301, 2020.

[3] J. Park and Y. Kim, "Supervised-learning-based optimal thermal management in an electric vehicle," *IEEE Access*, vol. 8, pp. 1290–1302, 2019.

[4] T. Linjordet and K. Balog, "Impact of training dataset size on neural answer selection models," in *European Conference on Information Retrieval*, pp. 828–835, Springer, 2019.

[5] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[10] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning (adaptive computation and machine learning series)," *Cambridge Massachusetts*, pp. 321–359, 2017.

[11] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," *Advances in neural information processing systems*, vol. 9, 1996.

[12] S.-C. Huang and Y.-F. Huang, "Bounds on number of hidden neurons of multilayer perceptrons in classification and recognition," in *IEEE International Symposium on Circuits and Systems*, pp. 2500–2503, IEEE, 1990.

[13] Y. Ito, "Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory," *Neural Networks*, vol. 4, no. 3, pp. 385–394, 1991.

[14] P. Sibi, S. A. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of theoretical and applied information technology*, vol. 47, no. 3, pp. 1264–1268, 2013.

[15] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[16] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural networks*, vol. 1, no. 4, pp. 295–307, 1988.

[17] J. Brownlee, "What is the difference between a batch and an epoch in a neural network," *Machine Learning Mastery*, vol. 20, 2018.

[18] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[19] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[20] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.