Microwave Heating Modelling

An application of neural network

How does the final temperature of a cup of water depends on the time microwaved and its initial volume of water?

1. Introduction

Microwaves are commonly used domestic appliances for heating. For me, I use it to heat milk everyday. Soon I noticed that if I fill the cup half full, and microwave it for half the time I used for a full cup, the milk would be too hot to drink. Therefore, I become curious about how the final temperature, the dependent variable, would be change according to different volumes and times microwaved, the two independent variables. But to avoid wasting milk, I decided to use water to explore this two-variable dynamic.

To start off, I collected the final temperatures of water with 10 different volumes and 10 different temperatures. Then I plotted the data on a 3-dimensional scatter plot. The scatter graph assembled a curved surface, and it would be very hard to fit a surface manually. In fact, I did not find software that can do the fitting for me. Based on my interest and experience in programming, I recalled how neural network, a modelling method, solved real-life problems by using the data to train itself to produce a function. The idea of creating a neural network made me intrigued as it allowed me to use a relatively "new" and interesting method and also to practice my Matlab programming skills.

2.Background

Matrix

Matrices are containers of data, which consists of rows (i) and columns (j) that may represent different quantities under different circumstances (Fig 2.1). Individual items (having a specific column and row number) in a matrix are called elements. In neural networks, matrices are very important not only because they act like data containers, but also due to the convenience matrix operations brought to us.

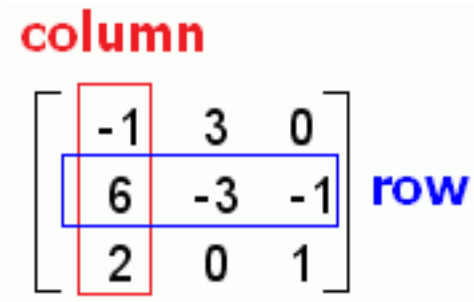


Figure 2.1 Column and Row of a Matrix

In matrix addition or subtraction, two matrices having the same dimension (same number of columns and rows) can be added or subtracted from each other element-wise, where $(A \pm B)_{i,j} = A_{i,j} \pm B_{i,j}, where\ 1 \leq i \leq m\ and\ 1 \leq j \leq n$, and this allows thousands if not millions of calculations to be done at the same time.

In matrix multiplication, the dot product, individual elements in the rows of the first matrix (m*n) are multiplied element-wise with the corresponding elements in the columns of the second matrix (n*k), and the sum of the products are placed at (i,j) where i equals the row number involved in calculation and j equals the column number involved in calculation:

$$[A\ B]_{i,j} = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \cdots + A_{i,n}B_{n,j} = \sum_{r=1}^{n} A_{i,r}B_{r,j}$$

In matrix transpose operation, the rows and columns of the matrix are set to the columns and rows respectively: $[A^T]_{i,j} = A_{j,i}$. The major reason for his operation is due to a practical reason that the multiplication of two matrices is much faster when one matrix is transposed beforehand in certain programming languages. Without matrix transpose, when the number of calculations is large, the processing time would be long.

In Hadamard product operation, two matrices with identical dimensions are multiplied element-wise: $(A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}$.

Neural Networks

Neuron networks come with different complexity. In my exploration, since only two input (independent) variables are used, a simple neuron network was used (Fig 2.1).

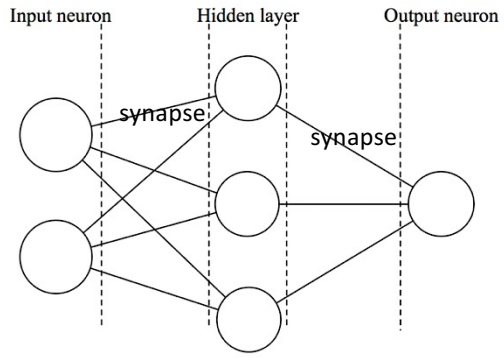


Figure 2.1 The Basic Construct of My Neural Network

The neuron network consists of three layers, the input neurons, the hidden layer and the output neuron. Data, which in this case are my volume and time microwaved, was first inputted through the 2 input neurons respectively, processed by synapses, and then delivered to the hidden layer. The hidden layer again processed the data from the synapses and output the processed data. Then, the data processed by hidden layer was processed by the synapses between the hidden layer and the output neuron. At last, the output neuron received the data from the synapses and output the final result.

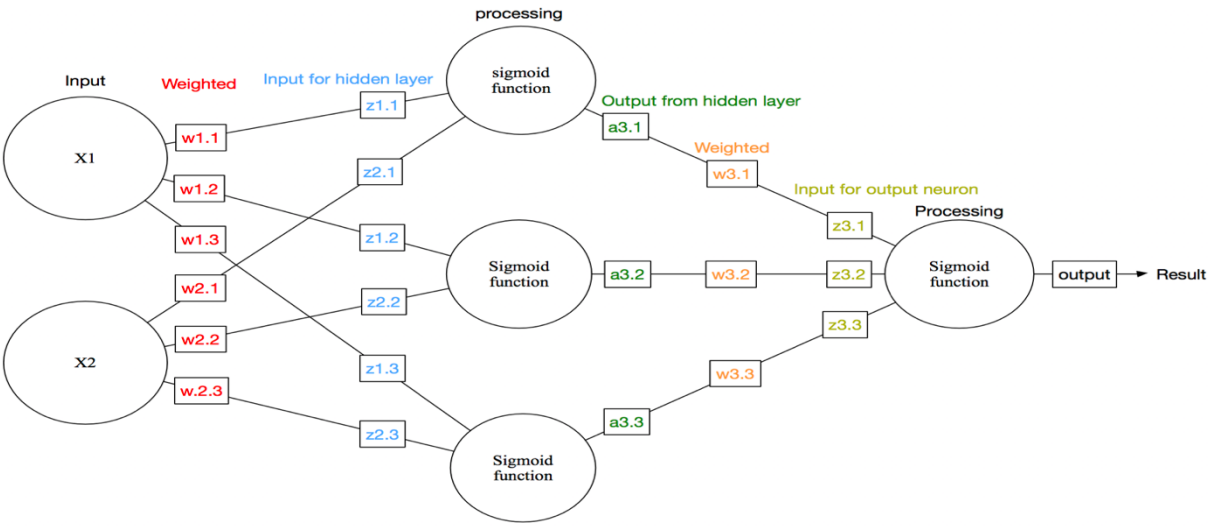


Figure 2.2

To illustrate how my neural network process data, we can first input a sample set of data:

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

A sample

The 2 columns in this matrix represents the two independent variables and the 3 rows represent the different combinations of them. This input matrix, X, can then multiply the first weight matrix, $W^{(1)}$, through dot product. $W^{(1)}$ represent the weighting between the input

Number of elements in a sample = rows of weights needed
Does not affect the number of neurons

neuron and the hidden layer. Matrix dot product is used to compute calculations involved in all neurons all at once to save computing time. Input values are marked orange.

$$W^{(1)} = \begin{bmatrix} w1.1 & w1.2 & w1.3 \\ w2.1 & w2.2 & w2.3 \end{bmatrix}$$

Number of neurons = # of columns in the weight matrix

$$XW^{(1)} = \begin{bmatrix} 1w1.1 + 2w2.1 & 1w1.2 + 2w2.2 & 1w1.3 + 2w2.3 \\ 3w1.1 + 4w2.1 & 3w1.2 + 4w2.2 & 3w1.3 + 4w2.3 \\ 5w1.1 + 6w2.1 & 5w1.2 + 6w2.2 & 5w1.3 + 6w2.3 \end{bmatrix} =$$
$$\begin{bmatrix} z1.1_{\,1} + z2.1_{2} & z1.2_{\,1} + z2.2_{2} & z1.3_{\,1} + z2.3_{2} \\ z1.1_{\,3} + z2.1_{4} & z1.2_{\,3} + z2.2_{4} & z1.3_{\,3} + z2.3_{4} \\ z1.1_{\,5} + z2.1_{6} & z1.2_{\,5} + z2.3_{6} & z1.1_{\,5} + z2.1_{6} \end{bmatrix}$$ (refer to fig 2.2)

Let us name the matrix $XW^{(1)}$ as $Z^{(2)}$, the input matrix for hidden layer. Each column represents the values inputted into each individual neuron.

Each neuron acts as a processing unit which all apply a function called the sigmoid function to each value in the "input matrix for hidden neuron". The sigmoid function (Figure 2.2) is a common activation function chosen in neural network applications because the shape of the curve assembles the graph of voltage for electrical impulses in the human brain (Figure 2.3).

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure 2.3 Sigmoid Function

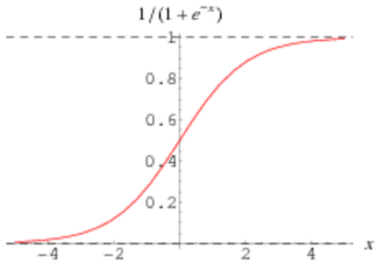


Figure 2.4 Shape of Sigmoid Function

Let us name the output matrix of all the hidden neurons $A^{(2)}$,

$$A^{(2)} = f\left(Z^{(2)}\right) = \begin{bmatrix} f(1w1.1 + 2w2.1) & f(1w1.2 + 2w2.2) & f(1w1.3 + 2w2.3) \\ f(3w1.1 + 4w2.1) & f(3w1.2 + 4w2.2) & f(3w1.3 + 4w2.3) \\ f(5w1.1 + 6w2.1) & f(5w1.2 + 6w2.2) & f(5w1.3 + 6w2.3) \end{bmatrix} =$$
$$\begin{bmatrix} a3.1_{1\ and\ 2} & a3.2_{1\ and\ 2} & a3.3_{1\ and\ 2} \\ a3.1_{3\ and\ 4} & a3.2_{3\ and\ 4} & a3.3_{3\ and\ 4} \\ a3.1_{5\ and\ 6} & a3.2_{5\ and\ 6} & a3.3_{5\ and\ 6} \end{bmatrix}$$ (refer to fig 2.2)

Again each column represents the values outputted by each individual neuron, since there are three neurons, there are three columns. Each row represents the calculation for each combination of independent variables.

Now $A^{(2)}$ is weighted again before being inputted into the output neuron, but with a different weight matrix called $W^{(2)}$. The dot product of $A^{(2)}$ and $W^{(2)}$ is called $Z^{(3)}$, the input for the output neuron.

$$W^{(2)} = \begin{matrix} w3.1 \\ [w3.2] \\ w3.3 \end{matrix}$$

$$Z^{(3)} = A^{(2)}W^{(2)} =$$
$$\begin{bmatrix} f(1w1.1 + 2w2.1) & f(1w1.2 + 2w2.2) & f(1w1.3 + 2w2.3) \\ f(3w1.1 + 4w2.1) & f(3w1.2 + 4w2.2) & f(3w1.3 + 4w2.3) \\ f(5w1.1 + 6w2.1) & f(5w1.2 + 6w2.2) & f(5w1.3 + 6w2.3) \end{bmatrix} \begin{bmatrix} w3.1 \\ w3.2 \\ w3.3 \end{bmatrix} =$$
$$\begin{bmatrix} w3.1f(1w1.1 + 2w2.1) + w3.2f(1w1.2 + 2w2.2) + w3.3f(1w1.3 + 2w2.3) \\ w3.1f(3w1.1 + 4w2.1) + w3.2f(3w1.1 + 4w2.1) + w3.3f(3w1.1 + 4w2.1) \\ w3.1f(5w1.1 + 6w2.1) + w3.2f(5w1.1 + 6w2.1) + w3.3f(5w1.1 + 6w2.1) \end{bmatrix} =$$
$$\begin{bmatrix} z3.1_{1\ and\ 2} + z3.2_{1\ and\ 2} + z3.2_{1\ and\ 2} \\ z3.1_{3\ and\ 4} + z3.2_{3\ and\ 4} + z3.2_{3\ and\ 4} \\ z3.1_{5\ and\ 6} + z3.2_{5\ and\ 6} + z3.2_{5\ and\ 6} \end{bmatrix}$$ (refer to fig 2.2)

$Z^{(3)}$ is then inputted into output neuron, which again apply a sigmoid function on $Z^{(3)}$ and output the final result, called $\hat{y}$, since it is the predicted value, not the actual values of the dependent variable.

$$\hat{y} = f\left(Z^{(3)}\right)$$
$$= \begin{bmatrix} f(w3.1f(1w1.1 + 2w2.1)) + f(w3.2f(1w1.2 + 2w2.2)) + f(w3.3f(1w1.3 + 2w2.3)) \\ f(w3.1f(3w1.1 + 4w2.1)) + f(w3.2f(3w1.1 + 4w2.1)) + f(w3.3f(3w1.1 + 4w2.1)) \\ f(w3.1f(5w1.1 + 6w2.1)) + f(w3.2f(5w1.1 + 6w2.1)) + f(w3.3f(5w1.1 + 6w2.1)) \end{bmatrix}$$

The final output matrix $\hat{y}$ has a dimension of 3*1. Compared to the input matrix, which has a dimension of 3*2, $\hat{y}$ has successfully computed the combinations of two independent variable values into corresponding one dependent variable value.

As we can see, the only things that can be varied during the calculation are the weights. And only specific combinations of weights could give us accurate predictions. This goal will be accomplished in the Calculations section.

3.Calculations

Before the creation of the model, the different volume of water (v) put into the microwave and different time microwaved (t) were recorded, and corresponding final temperature of water (temp) was recorded as well. Data collection table can be found in Appendix.

The data collected was first put into a matrix with 100 rows and 2 columns with the first column representing v and second column representing t. This data-collection matrix is named capital X:

$$X = \begin{bmatrix} 25 & 14 \\ \vdots & \vdots \\ 250 & 140 \end{bmatrix}$$

Since sigmoid function (Fig 2.4) only has a large gradient between the x-value between -2 and 2, the data I collected need to be normalized as small values so that they have a larger difference after sigmoid functions are applied to them. There are many ways to normalize data, but in this simple case, each v value is divided by the maximum v value and each t value

is divided by the maximum t value. From the Appendix, we can know that the maximum value of v is 250 and the maximum value of t is 140, thus, we can calculate the Hadamard product of:

$$X_{normalized} = X \circ X_{normalization\ matrix} = \begin{bmatrix} 25 & 14 \\ \vdots & \vdots \\ 250 & 140 \end{bmatrix} \circ \begin{bmatrix} \frac{1}{250} & \frac{1}{140} \\ \vdots & \vdots \\ \frac{1}{250} & \frac{1}{140} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix}$$

Multiply $X_{normalized}$ with the first weight matrix, $W^{(1)}$, through dot product will result in the input matrix, $Z^{(2)}$, for the hidden neurons. Let us assume that all the weights in the $W^{(1)}$ matrix have the value 0.5.

$$Z^{(2)} = X_{normalized}W^{(1)} = \begin{bmatrix} 0.1 & 0.1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ \vdots & \vdots & \vdots \\ 1 & 1 & 1 \end{bmatrix}$$

When the each hidden neuron receive $Z^{(2)}$ as input, a sigmoid function is applied to each element of $Z^{(2)}$, which result in the output matrix from the hidden neurons, $A^{(2)}$:

$$A^{(2)} = f\left(Z^{(2)}\right) = \begin{bmatrix} f(0.1) & f(0.1) & f(0.1) \\ \vdots & \vdots & \vdots \\ f(1) & f(1) & f(1) \end{bmatrix} = \begin{bmatrix} 0.5250 & 0.5250 & 0.5250 \\ \vdots & \vdots & \vdots \\ 0.7311 & 0.7311 & 0.7311 \end{bmatrix}$$

$A^{(2)}$ is then weighted by multiplying the second weight matrix, $W^{(2)}$, resulting in the input matrix for the output neuron $Z^{(3)}$. Let us assume that all the weights in the $W^{(2)}$ matrix have the value 0.5.

$$Z^{(3)} = A^{(2)}W^{(2)} = \begin{bmatrix} 0.5250 & 0.5250 & 0.5250 \\ \vdots & \vdots & \vdots \\ 0.7311 & 0.7311 & 0.7311 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.7875 \\ \vdots \\ 1.0966 \end{bmatrix}$$

The output neuron again apply the sigmoid function to each of the element in $Z^{(3)}$, and output the final output matrix, $\hat{y}$ $(y\ hat)$, the prediction of the final temperatures.

$$\hat{y} = f\left(Z^{(3)}\right) = \begin{bmatrix} f(0.7875) \\ \vdots \\ f(1.0966) \end{bmatrix} = \begin{bmatrix} 0.6873 \\ \vdots \\ 0.7496 \end{bmatrix}$$

Temp matrix is then normalized to the actual output matrix, y,

$$y = temp \circ temp_{normalization\ matrix} = \begin{bmatrix} 58 \\ \vdots \\ 52 \end{bmatrix} \begin{bmatrix} \frac{1}{100} \end{bmatrix} = \begin{bmatrix} 0.58 \\ \vdots \\ 0.52 \end{bmatrix}$$

The difference between the actual output, $temp_{normalized}$, and the predicted output by the neural network, $\hat{y}$, is obvious. This difference (or error) is defined using a half squared error cost function, J, where the sigma notation sums up the squared differences between corresponding elements of y anb $\hat{y}$ after matrix subtraction:

$$J = \frac{1}{2}\sum((y-\hat{y})^2) = \frac{1}{2}\sum(\begin{pmatrix} 0.58 \\ \vdots \\ 0.52 \end{pmatrix} - \begin{pmatrix} 0.6873 \\ \vdots \\ 0.7496 \end{pmatrix}\text{^}2)$$

$$= \frac{1}{2}\sum\begin{pmatrix} (-0.1073)^2 \\ \vdots \\ (-0.2296)^2 \end{pmatrix} = \frac{1}{2} * 8.0663 = 4.0331$$

The computed error here seems to be small, but this value, 4.0331, has a high order of magnitude than the normalized data. Therefore, this error needs to be reduced by having a more optimal combination of the 9 weights involved in weight matrices, $W^{(1)}$ and $W^{(2)}$. Substituting random numbers into the 9 weights in hope of finding the best combination "brutally" might seem to be feasible, however, if there are 1000 different values a weight can be, 10^{27} calculations have to be done before all 9 weights are correctly found, which is non-realistic using personal computers. Therefore, we need to solve the problem using a different approach. Calculating the partial derivative of J with respect to $W^{(1)}$ and $W^{(2)}$ can save a lot of calculation time by telling us whether to increase or decrease individual weights:

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\frac{1}{2}\sum((y-\hat{y})^{\hat{2}})}{\partial W^{(1)}} = (y-\hat{y})\left(-\frac{\partial \hat{y}}{\partial W^{(1)}}\right) = -(y-\hat{y})\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right)\left(\frac{\partial Z^{(3)}}{\partial W^{(1)}}\right)$$
$$= -(y-\hat{y})\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right)\left(\frac{\partial Z^{(3)}}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial W^{(1)}}\right)$$

(The summation sign disappears as soon as we obtained $\frac{\partial \hat{y}}{\partial W^{(1)}}$, since the partial derivative of $\hat{y}$ is already considering every element of $W^{(1)}$, the summation sign is unneccessary)
Since $\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right)$ is merely the derivative of the sigmoid function, it can be replaced by $f'(Z^{(3)})$.

$$\hat{y} = f\left(Z^{(3)}\right) = \frac{1}{1+e^{-Z^{(3)}}}$$
$$\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right) = f'\left(Z^{(3)}\right) = \frac{e^{-Z^{(3)}}}{\left(1+e^{-Z^{(3)}}\right)^2}$$

It is also easy to spot that $\left(\frac{\partial Z^{(3)}}{\partial A^{(2)}}\right) = W^{(2)}$ and $\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right) = f'(Z^{(2)})$ and $\left(\frac{\partial Z^{(2)}}{\partial W^{(1)}}\right) = X_{nomalized}$:

$$-(y-\hat{y})\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right)\left(\frac{\partial Z^{(3)}}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial W^{(1)}}\right) =$$
$$= -(y-\hat{y})\left(f'\left(Z^{(3)}\right)\right)\left(\frac{\partial Z^{(3)}}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial W^{(1)}}\right)$$
$$= -(y-\hat{y})\left(f'\left(Z^{(3)}\right)\right)W^{(2)}f'\left(Z^{(2)}\right)X$$
$$= -(y-\hat{y})\left(f'\left(XW^{(1)}\right)W^{(2)}\right)W^{(2)}f'\left(XW^{(1)}\right)X$$

This result is then reorganised for matrix operations:

$$\frac{\partial J}{\partial W^{(1)}} = X^T(((-(y-\hat{y}) \circ \left(f'\left(XW^{(1)}\right)W^{(2)}\right)W^{(2)^T}) \circ f'\left(XW^{(1)}\right)$$

Then let us find the partial derivative of J with respect to $W^{(2)}$:

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\frac{1}{2}\sum((y-\hat{y})^{\hat{2}})}{\partial W^{(2)}} = -(y-\hat{y})\left(\frac{\partial \hat{y}}{\partial W^{(2)}}\right) = -(y-\hat{y})\left(\frac{\partial \hat{y}}{\partial Z^{(3)}}\right)\left(\frac{\partial Z^{(3)}}{\partial W^{(2)}}\right)$$
$$= -(y-\hat{y})\left(f'\left(f\left(XW^{(1)}\right)W^{(2)}\right)\right)f\left(XW^{(1)}\right)$$

This result is then reorganised for matrix operations:

$$\frac{\partial J}{\partial W^{(2)}} == f\left(XW^{(1)}\right)^T(-(y-\hat{y}) \circ \left(f'\left(f\left(XW^{(1)}\right)W^{(2)}\right)\right))$$

The weight updating formula (Berwick) is a rather a commonly used formula in the study of neural networks, where

$$W^{(1)}{}_{updated} = W^{(1)}{}_{original} - \frac{1}{J} * \frac{\partial J}{\partial W^{(1)}}$$
$$W^{(2)}{}_{updated} = W^{(2)}{}_{original} - \frac{1}{J} * \frac{\partial J}{\partial W^{(2)}}$$

After the weight updating, the two updated weights are put back into calculations again, which will result in new partial derivatives, and the process continue until the error, J, is reasonably small. 10000 iterations were done through Matlab, and the code of the program is placed in the Appendix.

The resulting $W^{(1)}$ and $W^{(2)}$:

$$W_{optimum}{}^{(1)} = \begin{bmatrix} 1.0127 & 1.0127 & -8.9806 \\ -6.0445 & -6.0445 & 5.0721 \end{bmatrix}$$

$$W_{optimum}{}^{(2)} = \begin{bmatrix} -0.7216 \\ -0.7216 \\ 2.8029 \end{bmatrix}$$

The resulting error, J, is 0.1741, which is still high compared to normalized data, but it is much lower than the initial 4.0331.

Say if we have two new inputs, x and y, then the input matrix would be [x y].

Just like when we first multiplied X by $W_{optimum}{}^{(1)}$, we now multiply [x y] with the new weight matrix:

$$Z_{x,y}{}^{(2)} = [x\ y]\begin{bmatrix} 1.0127 & 1.0127 & -8.9806 \\ -6.0445 & -6.0445 & 5.0721 \end{bmatrix}$$
$$= [1.0127x - 6.0445y \quad 1.0127x - 6.0445y \quad -8.9806x + 5.0721y]$$

Then, when we input this result into the hidden layer, the neurons would then apply the sigmoid function on each individual element:

$$A_{x,y}{}^{(2)} = f\left(Z_{x,y}{}^{(2)}\right)$$
$$= \left[\frac{1}{1+e^{-(1.0127x-6.0445y)}} \quad \frac{1}{1+e^{-(1.0127x-6.0445y)}} \quad \frac{1}{1+e^{-8.9806x+5.0721y}}\right]$$

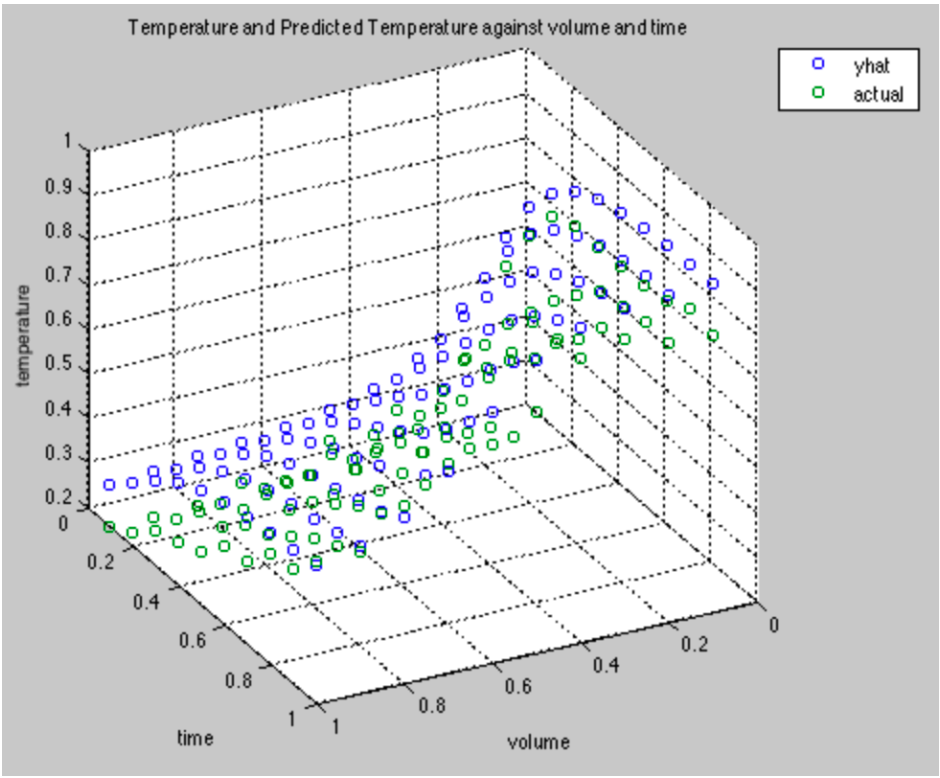
Then $A_{x,y}{}^{(2)}$ is outputted from the hidden layer and weighted by multiplying with $W_{optimum}{}^{(2)}$ to form the input matrix for the output neuron, $Z_{x,y}{}^{(3)}$,

$$Z_{x,y}{}^{(3)} =$$
$$= \left[\frac{1}{1+e^{-(1.0127x-6.0445y)}} \quad \frac{1}{1+e^{-(1.0127x-6.0445y)}} \quad \frac{1}{1+e^{-8.9806x+5.0721y}}\right]\begin{bmatrix}-0.7216\\-0.7216\\2.8029\end{bmatrix}$$
$$= \frac{-0.7216}{1+e^{-(1.0127x-6.0445y)}} + \frac{-0.7216}{1+e^{-(1.0127x-6.0445y)}} + \frac{2.8029}{1+e^{-8.9806x+5.0721y}}$$
$$= \frac{-1.4432}{1+e^{-(1.0127x-6.0445y)}} + \frac{2.8029}{1+e^{-8.9806x+5.0721y}}$$

At last, the output neuron again applies a sigmoid function on this result,

$$\hat{y} = \frac{1}{1+e^{\frac{1.4432}{1+e^{-(1.0127x-6.0445y)}} - \frac{2.8029}{1+e^{-(-8.9806x+5.0721y)}}}}$$

When the $\hat{y}$ is scatter-plotted against v and t in a 3-dimensional space using Matlab, the resulted surface is very close to the actual experimental result (Fig):



4.Evaluation of the results

Despite the fact that the algorithm of my neural network is able to produce a reasonably close model, there is no reason why the every dot on the predicted scatter plot appears to be higher than the corresponding actual data points graphed on the same scatter plot. Also, the function obtained which links v and t with predicted is too complicated to give us any understanding of the microwave heating phenomenon in terms of physics.

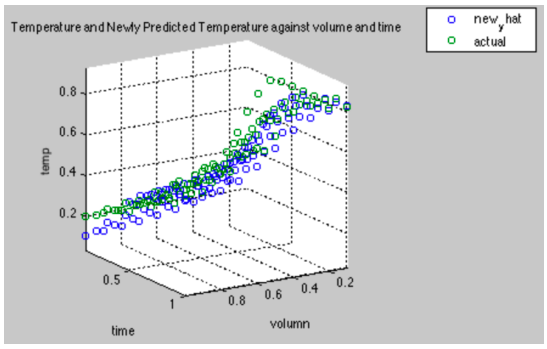


Figure 4.1 The New Predicted Temperature plotted against volume and time

The neural network contained too many random initial parameters, such as the value of 0.5 I chose for the weights. When I was exploring the neural network, I tried to vary the initial values of the weights, and I was shocked that different combinations of initial weights would give rise to different final weights. Since there is no best way to select the weights, I followed a suggestion in a online forum and randomly generated the weights in the first initial weight matrix according Gauss Distribution and with a mean of zero. I also did the same for the second initial weight matrix. Surprisingly, the resulting weight matrices were different from the onces generated from weight matrices containing 0.5. More interestingly, eventhough the resulting error, J, was larger when weights were randomly generated, the resulting function seem to have a better fit (Fig 4.1).

By researching on the internet, I found that weights should be tried out randomly so that I can actually find the *global minimum* on the cost function, J. Otherwise, only one minimum can be found and we cannot be sure whether it is global (fig 4.2). However, the global minimum can only be found by chance with random generated weights since the cost function with respect to each of the 9 individual weight would produce a 10 dimensional relationship, which is very hard to be plotted on a graph, not mentioning finding the global minimum of the graph. Only an analogy can be produced. As random initial weights would produce different partial derivatives of J with respect to $W^{(1)}$ and $W^{(2)}$, the weight updating functions would lead us into different minimums on the cost function, J, which give us bigger opportunities to find out the global minimum.

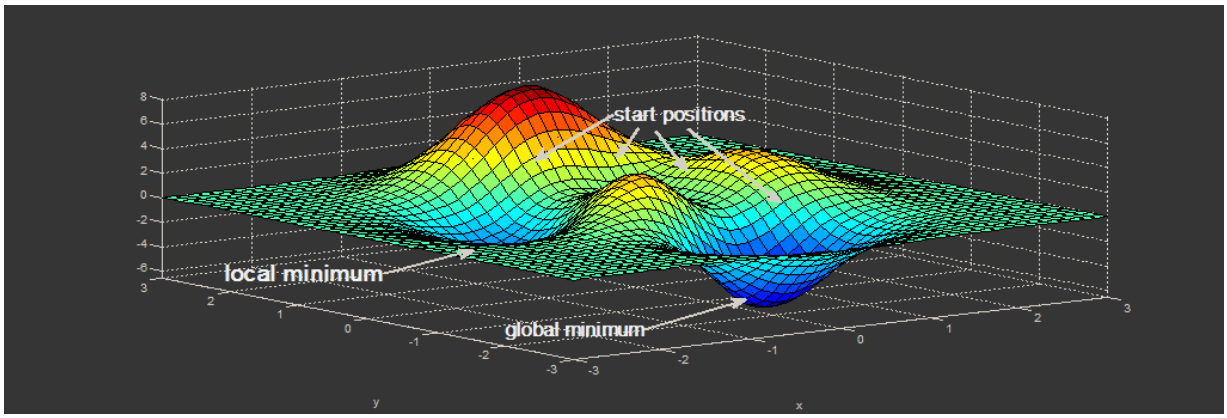


Figure 4.2 Analogy of finding the global minimum of the cost function, J

5.Conclusion

The aim of this mathematical exploration is to the final temperature of water based on the initial volume and time microwaved. From the equation I derived, I inputted initial volume as 65 ml and time microwaved as 75s, and the model outputted 56.6 degrees celcius, which is only slightly less than the actual final temperature of the water. Thus the model have successfully accomplished my goal and produced a valuble function for heating almost all water-based liquids at room temperature.

Work cited:

Wikipedia contributors. "Matrix (mathematics)." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 17 Apr. 2017. Web. 27 Apr. 2017.

Wikipedia contributors. "Matrix addition." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 14 Feb. 2017. Web. 27 Apr. 2017.

Wikipedia contributors. "Dot product." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 28 Mar. 2017. Web. 27 Apr. 2017.

Wikipedia contributors. "Transpose." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 20 Apr. 2017. Web. 27 Apr. 2017.

Welch Lab. “Neural Networks Demystified.” YouTube, uploaded by Welch Lab, 4 November 2014,
https://www.youtube.com/watch?v=bxe2T-V8XRs

“Backpropagation Algorithm” Unsupervised Feature Learning and Deep Learning. Stanford, 7 Appr. 2013. Web. 27 Apr. 2017,
http://deeplearning.stanford.edu/wiki/index/php/Backpropagation_Algorithm

Matlab Code

sigmoid.m (sigmoid function)

```
function y = sigmoid (x)
y = 1./(1+2.7182818285.^(-x);
```

sigmoidprime.m (derivative of sigmoid function)

```
function y = sigmoidprime (x)
y = (2.7182818285.^x)./((1+(2.7182818285.^x)).^2);
```

neural_network.m (code for the neural network that will output $W^{(1)}$ and $W^{(2)}$)

```
W1 = [0.5 0.5 0.5;0.5 0.5 0.5]; %
W2 = [0.5;0.5;0.5];

input = [v t];

X = input;

for n = 1:10000
   Z2 = X*W1;
   A2 = sigmoid (Z2);
   Z3 = A2 * W2;
   yhat = sigmoid (Z3);
   J = 0.5*(sum((temp-yhat).^2));

   dJ_dW1 = (X.')*((((-(temp-
yhat)).*sigmoidprime(sigmoid(X*W1)*W2))*(W2.')).*sigmoidprime(X*W1));
   dJ_dW2 = ((sigmoid(X*W1)).')*((-(temp-yhat)).*sigmoidprime(sigmoid(X*W1)*W2));

   W1 = W1 - (1/J).* (dJ_dW1);
   W2 = W2 - (1/J).* (dJ_dW2);

end
   disp(J)
   disp(W1)
   disp(W2)
```

Appendix

v	t	temp
25	14	58
25	28	69
25	42	78
25	56	80
25	70	80
25	84	80
25	98	80
25	112	81
25	126	84
25	140	82
50	14	30
50	28	47
50	42	56
50	56	63
50	70	70
50	84	76
50	98	81
50	112	82
50	126	83
50	140	83
75	14	26
75	28	35
75	42	50
75	56	53
75	70	59
75	84	72
75	98	72
75	112	77
75	126	84
75	140	86
100	14	27
100	28	37
100	42	40
100	56	50
100	70	62
100	84	62
100	98	72
100	112	75
100	126	84
100	140	84

v	t	temp
125	14	28
125	28	28
125	42	34
125	56	43
125	70	43
125	84	53
125	98	56
125	112	62
125	126	64
125	140	74
150	14	21
150	28	27
150	42	36
150	56	37
150	70	46
150	84	52
150	98	54
150	112	62
150	126	64
150	140	69
175	14	23
175	28	26
175	42	32
175	56	38
175	70	38
175	84	48
175	98	48
175	112	56
175	126	58
175	140	67
200	14	20
200	28	25
200	42	28
200	56	36
200	70	37
200	84	43
200	98	47
200	112	51
200	126	54
200	140	60

v	t	temp
225	14	20
225	28	24
225	42	30
225	56	31
225	70	36
225	84	38
225	98	42
225	112	46
225	126	49
225	140	52
250	14	20
250	28	23
250	42	28
250	56	30
250	70	32
250	84	38
250	98	39
250	112	45
250	126	46
250	140	52