Steam Table Prediction Using Artificial Neural Network

Jonathan Kirchman (B00610074)
Department of Chemical Engineering
Dalhousie University, Halifax, NS B3H 4R2
E-mail: jn690119@dal.ca

Abstract – *The motivation of this paper is to determine an artificial neural network to fit to highly nonlinear steam table values, thus saving computational memory and increasing speed.*

The artificial neural network is a multi-layer perceptron, utilizing gradient descent backpropagation. A logistic sigmoid transfer function was used for each layer, and the data was preprocessed to be between 0.1 and 0.9.

The artificial neural network was not completely successful. While the simulation ran, the error was quite high. The mean error of subcooled values of V, U, H and S were 9%, 48.6%, 49.7% and 24.5%, respectively. Clearly, more work needs to be completed before the neural network is truly fit to the data.

Secondly, a MatLab toolbox was implemented, which yielded excellent results. The network was trained to less than 1% error and output times were approximately 0.087 seconds.

Keywords: Artificial neural networks, perceptron, steam table, chemical engineering, heat transfer

1. INTRODUCTION

The motivation of this paper stems from a desire to save computational memory and increase computing speed when performing exceptionally long calculations. Storing the values and performing linear interpolation is memory intensive, and the polynomials are degree three or higher.[2]

Artificial neural networks (ANNs) are a potential solution to this problem, as once a network is trained, the only further computation required is a set of matrices. Perceptrons are based upon the human brain, which also learns through error correction.[3]

Therefore, the aim of this paper is to create an ANN that fits the data. Gradient descent backpropagation was chosen, as this is a relatively standard method. Logistic sigmoid functions will be used. It should be noted that, due to time constraints, only subcooled, ANN 1, was performed.

2. MODEL FORMULATION

2.1. Model Formulation

Steam tables are organized into subcooled, saturated and superheated values. These inputs are organized into an Excel sheet titled 'Steam_Tables.xlsx', which were then imported into MATLAB. Seven groups were made, shown in Table 1.

Table 1: Division of ANNs.

No.	Type	P (bar)
1	**Subcooled**	**0.01-150**
2	**Subcooled**	**200-373**
3	**Saturated**	**0.01-373**
4	**Superheated**	**0.01-1**
5	**Superheated**	**1-94**
6	**Superheated**	**94-220**
7	**Superheated**	**220-1000**

2.2. Numerical Methodology

A generic feed-forward neural network.

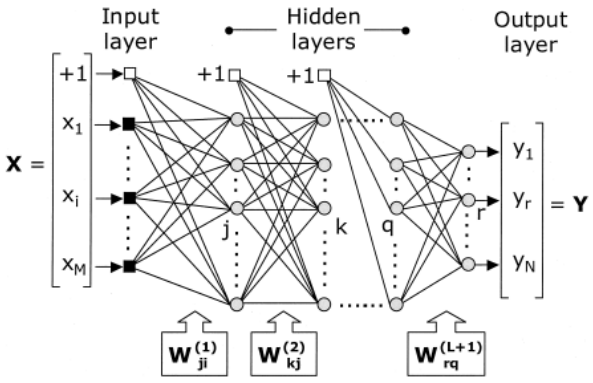

Figure 1: Feed-forward neural network. Accommodates multiple hidden layers, weights, inputs and outputs.[1]

The ANN for each network is shown in Fig. 2.

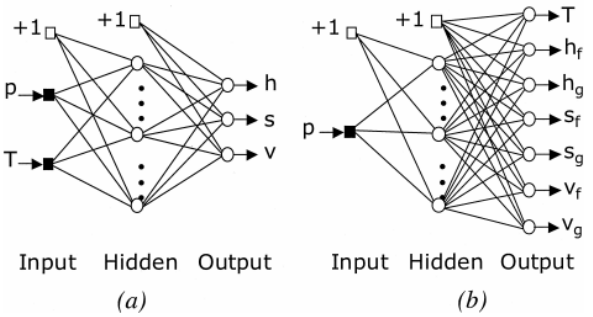

Fig. 2: ANNs for a) subcooled and b)saturated. Note that the hidden layers is more than one. As well, note that both the subcooled and saturated have internal energy, U, as an output.[1]

Each node has an analytic function, given by Eq. (1), with the transform of a shown in Eq. (2).

$$a_j = \sum_{i=0}^{d} w_{ji}^{1} * x_i \tag{1}$$

$$z_j = g(a_j) \tag{2}$$

Where: a is the analytic function of layer *j*, $w_{j\,i}$ is the weight, x_i is the input, g represents the activation function and z_j is the activated output of a.

Sigmoidal transfer functions were used, given by Eq. (3)

$$g(a_j) = \frac{1}{1+exp(-a_j)} \tag{3}$$

Derived from Bayes Theorem, sigmoidal transfer functions give outputs in the range of 0 to 1. This allows data sets with large fluctuations to be The output is given by Eq. (4).

$$y_k = g(\sum_{j=0}^{M} w_{kj}^{2} g(\sum_{i=0}^{d} w_{ji}^{1} * x_i)) \tag{4}$$

Where: y_k is the output.

At each node, the weights multiplied by the inputs must be summed, prior to activation. Error backpropagation is then performed. The first step is to find the error between the outputs and the expected values, shown in Eq. (5).

$$\delta_K = (y_k - t_K) \tag{5}$$

Where: δ_K is the difference between the outputs and expected values, y_k is the outputs, and t_K is the expected outputs.

The error function of layers, not including the output layer, is shown in Eq. (6).

$$\delta_j = z_j(1 - z_j) \sum_{k=1}^{c} w_{kj}\delta_k \tag{6}$$

Where: δ_j is any error not including the output layer. The multiplication of z_j by $(1-z_j)$ returns the derivative of z_j. That is, it deactivates the transfer function.

The overall error function is a standard sum-of-squares error function, given in Eq. (7).

$$E^n = \frac{1}{2} \sum_{k=1}^{c} (y_k - t_k)^2 \tag{7}$$

The derivatives were then taken with respect to their weights, shown in Eq. (8).

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j x_i \qquad \frac{\partial E^n}{\partial w_{kj}} = \delta_k z_j \tag{8}$$

Given a fixed-step gradient descent technique, Eq.(8) reduces to Eq. (9).

$$\Delta w_{ji} = -\eta \sum_n \delta_j^n x_i^n \tag{9}$$

Where: η is the learning rate of the function. Addition of a momentum term gives Eq. (10)

$$\Delta w = -\ \eta \Lambda E|_{w^{\tau}} + \mu \Delta w^{(\tau-1)} \quad (10)$$

Where: μ is the momentum rate and τis the step number. Note: in the simulation, τis given by `update`.

The momentum term effectively smooths oscillations throughout the data. As well, the momentum term serves to increase the learning rate, allowing for faster convergence with minimal divergent oscillations.

Once Eq. (9) and (10) are calculated, the Δw is applied to the weights, and the iteration is performed again.

2.3. Subcooled

Subcooled values of the steam table were chosen for this study. As Table 1 states, the values were divided by pressure, with ANN1 being 0.01 to 150 bar and ANN2 being 200 to 373 bar. Similar analysis was applied to each ANN.

The data was imported with inputs and outputs known. The inputs were then preprocessed by Eq. (11).

$$X1 = \frac{r_a - r_b}{b-a} * (X_i - a) + rb \quad (11)$$

Where: X1 is the normalized inputs, r_a is 0.9, r_b is 0.1, a is minimum value of all inputs in that row (ie. minimum temperature input), b is the maximum values of all inputs in a row. Note: in 'Steam_Tables.xlsx' the inputs are given as columns, however they are transposed upon import.

The learning rate was set at 0.03, as larger values exacerbated the oscillations, producing values of infinity and `NaN`. The network was initially composed for four weights and one hidden layer, however it was modified to be weight and layer adjustable. 15 weights and 5 hidden layers were chosen for this network.

As previously stated, logistic sigmoid transfer functions were used, which produced values between 0 and 1. Therefore, once y_k was reached, it was multiplied by the maximum known value in its row.

While momentum was attempted to be accounted for, there seemed to be a portion of the code at which the values were reduced to zero. However, parts of the implementation are left as comments in the code.

Training was performed through multiple iterative procedures. The first was to randomize the inputs, to reduce 'trend-finding' of the weights. In other words, this increases the weights ability to extrapolate to further data, as they are more generalized. This was done using `randperm` which provides a sufficient randomization.

The second aspect of training was to update the weights for each permutation, multiple times. As the learning rate was only 0.03, this allowed the weights to adjust to the inputs and further train themselves.

The inputs were portioned between supervised and unsupervised training sections. The supervised training iterated the same data and updated the weights, while the unsupervised training section only calculated y_k. The supervised and unsupervised sections contain identical code up until the output of y_k.

The iterations were performed for a total of 110 permutations of the input data. The number of updates each permutations went through iteratively decreased. This was done by setting `update = 1:nump` where `nump` is the number of total permutations over the current permutation. This allows for the last update, `update = 1:nump,`to only update once. This reduces dependency on iterative updating to correct error from the previous iteration.

2.4. Toolbox

The neural network MatLab toolbox was utilized. The `feedforwardnet` function only uses 1 hidden layer and N amount of neurons. Above, the hidden layers were increased to increase accuracy. However 1 hidden layer is sufficient when using this toolbox.

`Levenberg-Marquardt` was implemented instead of gradient descent. During training, the toolbox displayed iterations, error and parameters.

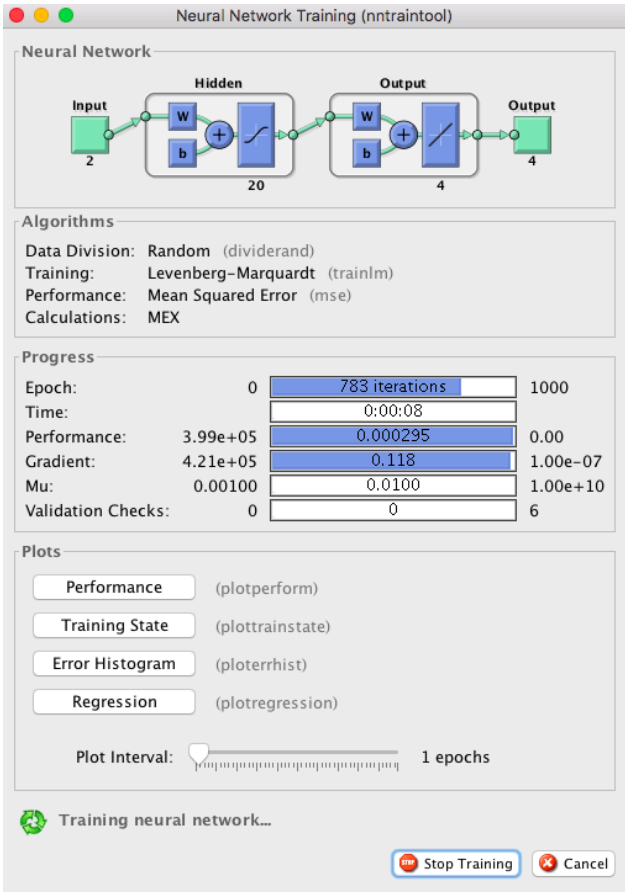

Figure 3: Example of toolbox interface during training. Epoch, time, performance, gradient, mu and validation checks are displayed.

3. RESULTS AND DISCUSSION

3.1. Artificial Neural Network

Table 2 displays the mean error percentage of each simulation.

Table 2: Mean error percentage

No.	Type	V (%)	U (%)	H (%)	S (%)
1	**Subcooled**	**8.24**	**65.01**	**68.13**	**43.06**
2	**Subcooled**	**11.20**	**32.15**	**31.24**	**5.88**

Clearly, the results are not ideal. The lowest errors occurred for volume and entropy, which have the lowest variations between numbers. For example, Vmin was 0.0009 and Vmax was 0.0015, whereas Hmin was 2.54 and Hmax was 1704. This makes it difficult for the network to predict values with such large variation.

The second network performed significantly better than the first. Likely, this is due to extreme nonlinearity in the first network, with reduced nonlinearity in the second.

3.2. Toolbox Network

To train the network, the same divisons seen in Table 1 were used. This resulted in 7 trained networks. Once trained, the Error Histogram was viewed.

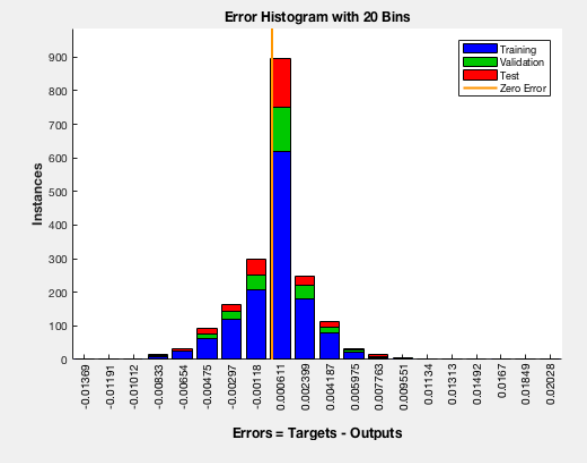

Figure 4: Error Histogram of training Net1.

As well, validation of performance was viewed.

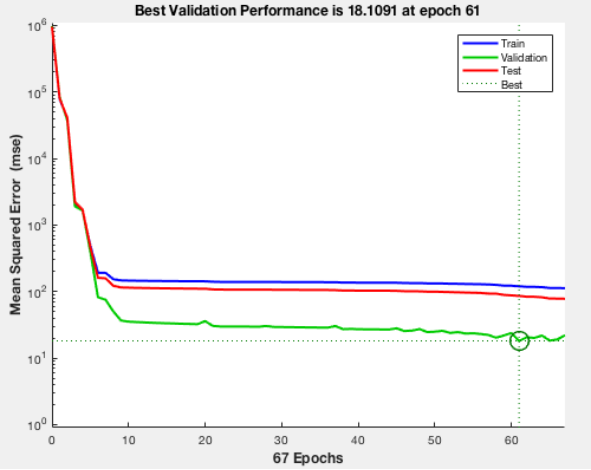

Figure 5: Training performance validation of Net1.

To test the toolbox, `Test(800,90)` was inputted into the network. 800 represents the temperature, in celsius, while 90 represents the pressure, in bar.

Results showed a significant improvement in the accuracy of the neural network. Outputs each had less than 1% error, and was calculated in a time of 0.0868 seconds. This is nearly the 20x faster than the polynomial

calculations that the original paper claimed.

4. CONCLUSIONS

Overall, an artificial neural network was created with 15 weights and 5 hidden layers. It accepted 2 inputs plus bias which it then transformed into 4 outputs. The error was high, likely due to the extreme nonlinear behaviour of steam table values, and therefore more work must be completed prior to implementation in codes.

Neural Network Toolbox from MatLab was implemented, with a significant improvement in results. The calculation time was 0.0868 seconds, 20x faster than the 1 second polynomial calculations that Aziman claimed. The accuracy was within 1%.

Nomenclature

Symbols

w	Weight
a	Analytical function
g	Transfer function
δ	Error
M	Number of inputs
N	Number of outputs

Subscripts

j	Hidden layer
k	Output layer
i	Input layer

Superscripts

τ	Step number

References

[1] A. R. Aziman, J. J. Arriagada, M. Assadi, "Generation of steam tables using artificial neural networks," *Journal of Heat Transfer Engineering*, vol. 25, no. 2, pp. 41-51, 2004

[2] W. J. Garland and B.J. Hand, "Simple functions for the fast approximation of light water thermodynamic properties," *Nuclear Engineering and Design*, vol. 113, pp. 21-34, 1989.
Available as of March 3, 2017 from https://canteach.candu.org/Content%20Library/20044002.pdf

[3] R. D. Seidler, Youngbin Kwak, B. W. Fling, J. A. Bernard, "Neurocognitive mechanisms of error-based motor learning," *Journal of Experimental Medicine and Biology*, vol. 782, pp. 1-21, 2013
Available as of March 3, 2017 from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3817858/pdf/nihms-521432.pdf

[4] Christopher. M. Bishop, *Neural NEtworks for Patter Recognition*, Birmingham, UK, Claredon Press, 1995, 498 pp.