Fundamentals of Backpropagation

June 2024

Joseph Lazarus, Ctr

To develop and apply independent, tailored Scientific Test & Analysis Techniques solutions to Test and Evaluation that deliver insight to inform better decisions.

About this Publication:
This work was conducted by the Scientific Test & Analysis Techniques Center of Excellence under contract FA8075-18-D-0002, Task FA8075-21-F-0074.

For more information:
Visit, www.AFIT.edu/STAT
Email, AFIT.ENS.STATCOE@us.af.mil
Call, 937-255-3636 x4736

Technical Reviewers:
Corinne Stafford
Nicholas Jones

Scientific Test & Analysis Techniques Center of Excellence
2950 Hobson Way
Wright-Patterson Air Force Base, Ohio

Version: 1, FY24

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) have significantly impacted numerous domains, yet the complexity of underlying algorithms often remains obscure to non-technical stakeholders, such as managers and executives and technical T&E practitioners. This paper clarifies one of the foundational algorithms in neural networks: backpropagation. It aims to elucidate the principles and mechanisms of backpropagation in accessible terms for the test and evaluation community, providing a comprehensive overview of AI and ML concepts and an introduction to neural networks (NNs). Historical context is provided through seminal works by Rumelhart (1986) and LeCun (1998). A practical demonstration using an Excel simulation illustrates backpropagation in a simple neural network, aiding managers in understanding the algorithm's functionality and enabling them to better evaluate and implement AI technologies within their organizations.

Keywords: backpropagation, machine learning, AI, neural networks

Table of Contents

Introduction

Artificial Intelligence (AI) and Machine Learning (ML), particularly Neural Networks (NNs), have demonstrated remarkable capabilities across various domains, from image and speech recognition to game playing and autonomous driving. However, the intricacies of how these technologies operate often remain opaque, especially to non-technical stakeholders such as managers and executives. The complexity of neural network architectures and their training algorithms can obscure understanding, making it challenging to evaluate their performance and potential applications. This paper aims to demystify one of the foundational algorithms in neural networks, backpropagation, to enhance the comprehension of ML concepts among business leaders.

Purpose

The primary purpose of this paper is to educate test and evaluation practitioners, managers, and executives about the fundamental mechanisms and principles underlying backpropagation. By breaking down this concept into more accessible terms and demonstrating their practical applications, this paper seeks to empower decision-makers with the knowledge to better assess and leverage AI technologies within their organizations.

Overview of Content

This paper will begin by providing background information on the basic concepts of AI and ML, including an introduction to neural networks (NNs). It will then delve into the backpropagation algorithm, explaining its purpose, basic functioning, and historical context, referencing seminal works by Rumelhart et al. (1986) and LeCun et al. (1998). Following this, the paper will explore how backpropagation is integrated into neural networks to enhance learning, including an explanation of the various layers and structures of NNs. A practical demonstration, available in accompanying video, will illustrate key concepts of backpropagation using a simple neural network in Excel. The discussion will focus on how understanding these algorithms can aid managers in evaluating relevant use cases and assessing performance. Finally, the paper will summarize the key points and provide a comprehensive list of references.

Background

Basic Concepts of Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a broad field encompassing various technologies aimed at enabling machines to perform tasks that typically require human intelligence. This includes areas such as natural language processing, robotics, and computer vision. Machine Learning (ML) is a subset of AI that focuses on the development of algorithms and statistical models that allow computers to learn from and make predictions or decisions based on data. Within ML, neural networks
(NNs) are a powerful method, inspired by the structure and function of the human brain.

Introduction to Neural Networks

Neural networks are computational models composed of interconnected layers of nodes, or neurons, that process input data to generate outputs. These networks consist of an input layer that receives the raw data, one or more hidden layers that perform intermediate computations and non-linear transformations, and an output layer that produces the final result. They are designed to recognize patterns and relationships in data through a process of learning, which

involves adjusting the weights of the connections between neurons to minimize the difference between the actual output and the desired output. Figure 1 shows an example of a neural network, including an input layer of nodes (x1, x2), a hidden layer of nodes (h11, h12, h13), and an output layer of nodes (y), with weighted connections between nodes. The learning process in neural networks is primarily driven by the backpropagation algorithm, which computes the gradient of the loss function with respect to the weights by the chain rule, propagating the error backward through the network.

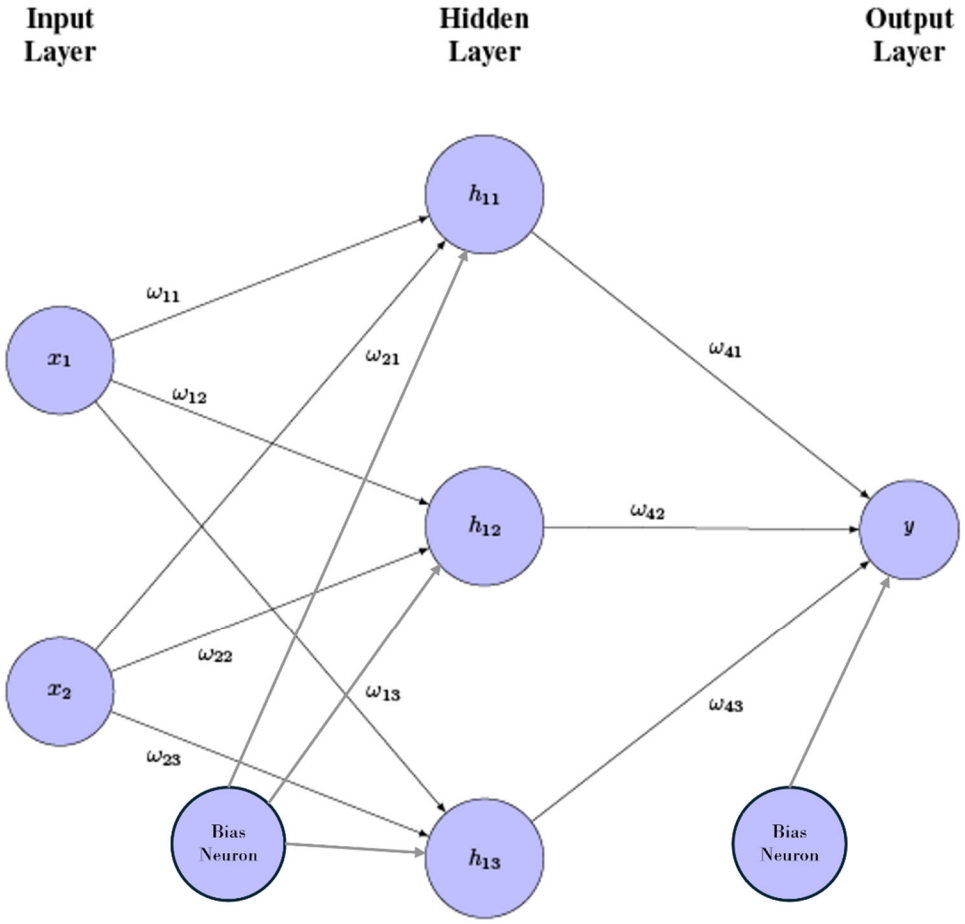

Figure 1
Example of a Neural Network

Understanding Backpropagation

Introduction to the Algorithm

Backpropagation, short for "backward propagation of errors," is the fundamental algorithm used for training neural networks. It is a supervised learning technique, which means it requires a set of input-output pairs to train the model. The primary objective of backpropagation is to minimize the error between the network's predictions and the actual target values. This is achieved by adjusting the weights and biases of the network in the direction that reduces the error. Backpropagation is a key component of the gradient descent optimization algorithm, which iteratively updates the network’s parameters by moving them in the direction of the negative

gradient of the loss function with respect to the weights. For an in-depth explanation of gradient descent, see “Fundamentals of Gradient Descent” (Lazarus, 2024). By combining backpropagation with gradient descent, the network effectively learns from the data, reducing the error and improving its predictive accuracy iteratively.

The backpropagation algorithm involves two main steps: the forward pass and the backward pass. During the forward pass, the input data is propagated through the network layer by layer to generate an output. This output is then compared to the target output, and the error is calculated. In the backward pass, this error is propagated back through the network, layer by layer, to update the weights and biases. This is done using the chain rule from calculus, which helps compute the gradient of the loss function with respect to each weight.

Historical Context

The backpropagation algorithm gained widespread recognition and use in the field of neural networks after the seminal work by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in 1986. Their paper, "Learning representations by back-propagating errors," provided a detailed explanation of the algorithm and demonstrated its effectiveness in training multi-layer neural networks. This work laid the foundation for the development of deep learning, where neural networks with many layers are trained to learn complex patterns in data.

The 1998 paper by LeCun et al., "Gradient-Based Learning Applied to Document Recognition," further advanced the field by applying backpropagation to Convolutional Neural Networks (CNNs). The work demonstrated the power of CNNs in image recognition tasks, showcasing the practical applications of backpropagation in real-world problems.

Mathematical Principles

The mathematical foundation of backpropagation is rooted in calculus, specifically the chain rule for derivatives. A derivative measures how a function changes as one of its inputs changes, while a gradient generalizes this concept for multiple inputs as a vector in the direction of the steepest ascent, where the elements of the vector are the partial derivatives with respect to each input. The chain rule allows us to decompose the derivative of a composite function into the product of the derivatives of its constituent functions. In the context of neural networks, this means that the gradient of the loss function can be expressed as the product of the partial derivatives with respect to the neurons, weights, and biases that compose the network.

Break Down of the Process

1. **Forward Pass**: Compute the output of the network for a given input by passing the input through each layer. Each layer applies a linear transformation (weighted sum plus bias) followed by a non-linear activation function.
    - **Linear Transformation:** The linear transformation for a neuron can be represented as:

      $z = w \cdot x + b$ where z is the net input to the neuron, w represents the weights, x is the input, and b is the bias.
    - **Bias:** The bias is a parameter in the neural network that is added to the weighted sum of inputs to the neuron.

- **Activation Function:** The activation function introduces non-linearity into the network, enabling it to learn more complex patterns. Common functions include the sigmoid functions, ReLU (Rectified Linear Unit), and tanh.

2. **Loss Calculation**: Compare the network's output to the target output and compute the loss (error). Common loss functions include Mean Squared Error (MSE) for regression tasks, which focus on predicting continuous values and Cross-Entropy Loss for classification tasks, where the focus is on predicting discrete labels or categories.
3. **Backward Pass**: Compute the gradient of the loss with respect to the weights and biases in the network. This involves:
    - **Output Layer**: Compute the gradient of the loss with respect to the output of the network.
    - **Hidden Layers**: Use the chain rule to propagate the gradient back through the network, layer by layer.
4. **Update Weights and Biases:** Adjust the weights and biases in the direction that minimizes the loss. This is done by subtracting the product of the learning rate (η) and the gradient from the current weights and biases. The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Mathematically, for a single layer, the update rule for a weight is given by:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

where eta (η) is the learning rate and $\frac{\partial L}{\partial w}$ is the partial derivative of the loss function L with respect to the weight w.

- **Learning Rate:** The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It is crucial to choose an appropriate learning rate; if it is too large, the model may not be able to arrive at the minimum and converge to a suboptimal solution, while if it is too small, the convergence process can be very slow.
    - Start with a small value. Common initial values range from 0.001 to 0.1
    - Use an adaptive method. Techniques like rate schedules or adaptive learning rate algorithms (e.g., Adam, RMSporp) can adjust the learning rate during training for better performance
    - Monitor convergence: Plot the loss function to ensure it decreases smoothly. If the loss fluctuates or diverges, the learning rate might be too high. If the plot decreases too slowly, the learning rate might be too low.

5. **Iterative Process and Stopping Conditions:** By following these steps, backpropagation iteratively adjusts the network parameters to minimize the loss, improving the model's accuracy over time. Steps 1-4 are repeated many times

(iterations) until the loss has been minimized. The optimization process typically stops when one or more of the following conditions are met:

- **Convergence**: The change in the gradient of the loss function between iterations falls below a predefined threshold.
- **Maximum Iterations**: A set number of iterations or epochs have been completed.
- **Early Stopping**: Stopping condition when the performance on the dataset starts to deteriorate, indicting potential overfitting.

The number of iterations required can vary significantly based on factors such as the complexity of the problem, the architecture of the network, and the size of the dataset. It is not uncommon for neural networks to go through thousands or even tens of thousands of iterations during training. For example, training a deep neural network on a large dataset like ImageNet might require tens of thousands of epochs, while simpler problems with smaller datasets might converge in just a few hundred of epochs.

Conclusion

Understanding the mechanics of backpropagation is essential for comprehending how neural networks learn and improve their performance. This paper has aimed to clarify the backpropagation algorithm by breaking down its fundamental principles, historical context, and mathematical underpinnings. By examining the forward and backward passes, loss calculation, and gradient descent, this paper aimed to provide a clear and accessible explanation of how backpropagation operates.

Backpropagation's significance in the field of machine learning cannot be overstated. It is the cornerstone of training neural networks, enabling them to adjust their weights and biases to minimize errors and enhance accuracy. The algorithm's introduction and subsequent development by pioneers like Rumelhart, Hinton, Williams, and LeCun have paved the way for the remarkable advancements in AI and deep learning that we witness today.

For test and evaluation managers and executives, gaining a solid understanding of backpropagation equips them with the knowledge to make informed decisions about adopting and implementing AI technologies within their organizations. This comprehension allows them to better evaluate the capabilities and limitations of neural networks, ensuring that they can leverage these powerful tools effectively to drive innovation and achieve T&E goals.

As AI and machine learning continue to evolve, the principles of backpropagation will remain foundational to the development of more sophisticated and capable neural networks. By demystifying this critical algorithm, we hope to empower test and evaluation community to harness the full potential of AI and machine learning, fostering a deeper appreciation of the technology's transformative impact on various industries.

To further illustrate the concepts discussed, an accompanying video provides a practical demonstration of backpropagation through a simple neural network implemented in Excel. This video offers a hands-on understanding of how backpropagation works in practice, highlighting the step-by-step process of training a neural network, adjusting the weights and biases, and

minimizing error to achieve accurate predictions. For those interested in building and running their own neural network, Appendix A: README for Simple Neural Network provides detailed instructions to build their own in Excel, Appendix B: Macro for Running Simple Neural Network in Excel includes the necessary VBA script to run the network.

References

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

Lazarus, J.I. (2024). Fundamental of Gradient Descent. Scientific Test and Analysis Techniques Center of Excellence (STAT COE).

Appendix A
README for Simple Neural Network

Backpropagation Implementation in Excel

This README provides a step-by-step guide to implementing a simple neural network with backpropagation in Excel. The example uses a neural network with one hidden layer for a binary classification problem.

Network Structure
- **Inputs:** 2 input neurons (`x1`, `x2`)
- **Hidden Layer:** 3 neurons (`h11`, `h12`, `h13`)
- **Output Layer:** 1 neuron (`y`)

Initialization
1. **Inputs and Target Output:**
   - `A1`: `x1` (Label for the first input)
   - `B1`: `x2` (Label for the second input)
   - `C1`: `y_true` (Label for the target output)
   - `A2`: Input value for `x1`
   - `B2`: Input value for `x2`
   - `C2`: Target output value

2. **Weights (initialized with random values):**
   - `F1`: `w11`
   - `F2`: `w12`
   - `F3`: `w13`
   - `F2`: `=RAND()` (Initial random value for `w11`)
   - `F3`: `=RAND()` (Initial random value for `w12`)
   - `F4`: `=RAND()` (Initial random value for `w13`)
   - `G1`: `w21`
   - `G2`: `=RAND()` (Initial random value for `w21`)
   - `G3`: `=RAND()` (Initial random value for `w22`)
   - `G4`: `=RAND()` (Initial random value for `w23`)
   - `H1`: `w31`
   - `H2`: `=RAND()` (Initial random value for `w31`)
   - `H3`: `=RAND()` (Initial random value for `w32`)
   - `H4`: `=RAND()` (Initial random value for `w33`)

3. **Biases:**
   - `I1`: `Bias`
   - `I2`: `=RAND()` (Initial random bias for the hidden layer)
   - `I3`: `=RAND()` (Initial random bias for the output layer)

4. **Learning Rate:**
   - `J1`: `Learning Rate`
   - `J2`: `0.1` (Learning rate value)

Forward Propagation
Calculate the output of the network based on current weights and biases.

Hidden Layer
- **Net Input to Hidden Neurons (`z11`, `z12`, `z13`):**
  - `K1`: `z11`
  - `K2`: `=D2*A2 + D3*B2 + H2`
  - `L1`: `z12`
  - `L2`: `=E2*A2 + E3*B2 + H2`
  - `M1`: `z13`
  - `M2`: `=F2*A2 + F3*B2 + H2`

- **Activation of Hidden Neurons (`h11`, `h12`, `h13`):**
  - `N1`: `h11`
  - `N2`: `=1 / (1 + EXP(-J2))`
  - `O1`: `h12`
  - `O2`: `=1 / (1 + EXP(-K2))`
  - `P1`: `h13`
  - `P2`: `=1 / (1 + EXP(-L2))`

Output Layer
- **Net Input to Output Neuron (`z2`):**
  - `Q1`: `z2`
  - `Q2`: `=G2*M2 + G3*N2 + G4*O2 + H3`

- **Activation of Output Neuron (`y`):**
  - `R1`: `y`
  - `R2`: `=1 / (1 + EXP(-P2))`

Error Calculation
Calculate the error of the network's output compared to the target output.

- **Error (`Error`):**
  - `S1`: `Error`
  - `S2`: `=-(C2*LN(Q2) + (1-C2)*LN(1-Q2))`

Backward Propagation of Errors
Calculate the gradients of the error with respect to the weights and biases.

Output Layer
- **Error Term for Output Neuron (`delta2`):**
  - `T1`: `delta2`
  - `T2`: `=Q2 - C2`

- **Gradients for Weights (`Delta w31`, `Delta w32`, `Delta w33`):**
  - `U1`: `Delta w31`
  - `U2`: `=S2*M2`
  - `V1`: `Delta w32`
  - `V2`: `=S2*N2`
  - `W1`: `Delta w33`
  - `W2`: `=S2*O2`

- **Gradient for Bias (`Delta b2`):**
  - `X1`: `Delta b2`

  - `X2`: `=S2`

Hidden Layer
- **Error Terms for Hidden Neurons (`delta11`, `delta12`, `delta13`):**
  - `Y1`: `delta11`
  - `Y2`: `=S2*G2*M2*(1-M2)`
  - `Z1`: `delta12`
  - `Z2`: `=S2*G3*N2*(1-N2)`
  - `AA1`: `delta13`
  - `AA2`: `=S2*G4*O2*(1-O2)`

- **Gradients for Weights (`Delta w11`, `Delta w12`, `Delta w13`, `Delta w21`, `Delta w22`, `Delta w23`):**
  - `AB1`: `Delta w11`
  - `AB2`: `=X2*A2`
  - `AC1`: `Delta w12`
  - `AC2`: `=Y2*A2`
  - `AD1`: `Delta w13`
  - `AD2`: `=Z2*A2`
  - `AE1`: `Delta w21`
  - `AE2`: `=X2*B2`
  - `AF1`: `Delta w22`
  - `AF2`: `=Y2*B2`
  - `AG1`: `Delta w23`
  - `AG2`: `=Z2*B2`

- **Gradient for Bias (`Delta b1`):**
  - `AH1`: `Delta b1`
  - `AH2`: `=X2 + Y2 + Z2`

Update Weights and Biases
Adjust the weights and biases using the gradients.

- `F7`: `=F2 - I2*AB2` (Updated weight `w11`)
- `F8`: `=F3 - I2*AC2` (Updated weight `w12`)
- `F9`: `=F4 - I2*AD2` (Updated weight `w13`)
- `H7`: `=G2 - I2*AE2` (Updated weight `w21`)
- `H8`: `=G3 - I2*AF2` (Updated weight `w22`)
- `H9`: `=G4 - I2*AG2` (Updated weight `w23`)
- `H7`: `=G2 - I2*U2` (Updated weight `w31`)
- `H8`: `=G3 - I2*V2` (Updated weight `w32`)
- `H9`: `=G4 - I2*W2` (Updated weight `w33`)
- `I8`: `=I2 - I2*AH2` (Updated bias for hidden layer)
- `I9`: `=I3 - I2*X2` (Updated bias for output layer)

Iteration
To train the network, manually copy the updated weights and biases back to their original cells (D2, E2, F2, etc.) after each iteration or use Excel’s iterative calculation feature. Repeat the steps for multiple epochs until the error converges.

Summary

1. **Forward Propagation:** Calculate activations.
2. **Error Calculation:** Compute the error.
3. **Backward Propagation:** Compute gradients.
4. **Update Weights and Biases:** Adjust parameters.

By iterating this process, the network learns to minimize the error and make accurate predictions.

Appendix B

Macro For Running Simple Neural Network in Excel

```
Sub IterateBackprop_reset_rand()
    Dim sheet As Worksheet
    Set sheet = ThisWorkbook.ActiveSheet

    ' Number of iterations
    Dim numIterations As Integer
    numIterations = 100

    ' Learning rate
    Dim eta As Double
    eta = sheet.Range("J2").Value

    Dim iter As Integer
    Dim errorArray() As Double
    ReDim errorArray(1 To numIterations)

    For iter = 1 To numIterations
        ' Forward propagation calculations
        Dim x1 As Double, x2 As Double, yTrue As Double
        Dim w11 As Double, w12 As Double, w13 As Double
        Dim w21 As Double, w22 As Double, w23 As Double
        Dim w31 As Double, w32 As Double, w33 As Double
        Dim b1 As Double, b2 As Double
        Dim z11 As Double, z12 As Double, z13 As Double
        Dim h11 As Double, h12 As Double, h13 As Double
        Dim z2 As Double, y As Double
        Dim delta2 As Double
        Dim deltaw31 As Double, deltaw32 As Double, deltaw33 As Double, deltaB2 As Double
        Dim delta11 As Double, delta12 As Double, delta13 As Double
        Dim deltaW11 As Double, deltaW12 As Double, deltaW13 As Double
        Dim deltaW21 As Double, deltaW22 As Double, deltaW23 As Double
        Dim deltaB1 As Double

        x1 = sheet.Range("A2").Value
        x2 = sheet.Range("B2").Value
        yTrue = sheet.Range("C2").Value

        w11 = sheet.Range("F2").Value
        w12 = sheet.Range("F3").Value
        w13 = sheet.Range("F4").Value
        w21 = sheet.Range("G2").Value
        w22 = sheet.Range("G3").Value
        w23 = sheet.Range("G4").Value
        w31 = sheet.Range("H2").Value
        w32 = sheet.Range("H3").Value
        w33 = sheet.Range("H4").Value

        b1 = sheet.Range("I2").Value
```

```
b2 = sheet.Range("I3").Value

z11 = w11 * x1 + w21 * x2 + b1
z12 = w12 * x1 + w22 * x2 + b1
z13 = w13 * x1 + w23 * x2 + b1

h11 = 1 / (1 + Exp(-z11))
h12 = 1 / (1 + Exp(-z12))
h13 = 1 / (1 + Exp(-z13))

z2 = w31 * h11 + w32 * h12 + w33 * h13 + b2
y = 1 / (1 + Exp(-z2))

' Calculate error and store it
errorArray(iter) = -((yTrue * Log(y)) + ((1 - yTrue) * Log(1 - y)))

' Backpropagation calculations
delta2 = y - yTrue

deltaw31 = delta2 * h11
deltaw32 = delta2 * h12
deltaw33 = delta2 * h13
deltaB2 = delta2

delta11 = delta2 * w31 * h11 * (1 - h11)
delta12 = delta2 * w32 * h12 * (1 - h12)
delta13 = delta2 * w33 * h13 * (1 - h13)

deltaW11 = delta11 * x1
deltaW12 = delta12 * x1
deltaW13 = delta13 * x1
deltaW21 = delta11 * x2
deltaW22 = delta12 * x2
deltaW23 = delta13 * x2
deltaB1 = delta11 + delta12 + delta13

' Update weights and biases
w11 = w11 - eta * deltaW11
w12 = w12 - eta * deltaW12
w13 = w13 - eta * deltaW13
w21 = w21 - eta * deltaW21
w22 = w22 - eta * deltaW22
w23 = w23 - eta * deltaW23
w31 = w31 - eta * deltaw31
w32 = w32 - eta * deltaw32
w33 = w33 - eta * deltaw33
b1 = b1 - eta * deltaB1
b2 = b2 - eta * deltaB2

' Set updated values back to sheet
sheet.Range("F2").Value = w11
```

```
        sheet.Range("F3").Value = w12
        sheet.Range("F4").Value = w13
        sheet.Range("G2").Value = w21
        sheet.Range("G3").Value = w22
        sheet.Range("G4").Value = w23
        sheet.Range("H2").Value = w31
        sheet.Range("H3").Value = w32
        sheet.Range("H4").Value = w33
        sheet.Range("I2").Value = b1
        sheet.Range("I3").Value = b2

        ' Delay for observation
        Application.Wait (Now + TimeValue("0:00:01")) ' Delay for 1 second

        ' Debug output
        Debug.Print "Iteration " & iter & ": Error = " & errorArray(iter)
    Next iter

    ' Write errors to the sheet for plotting
    sheet.Range("O10").Value = "Iteration"
    sheet.Range("P10").Value = "Error"
    Dim i As Integer
    For i = 1 To numIterations
        sheet.Range("O" & i + 10).Value = i
        sheet.Range("P" & i + 10).Value = errorArray(i)
    Next i

    ' Create a chart for the learning curve
    Dim chartObj As ChartObject
    Set chartObj = sheet.ChartObjects.Add(Left:=300, Width:=500, Top:=10, Height:=300)
    Dim chart As chart
    Set chart = chartObj.chart
    chart.SetSourceData Source:=sheet.Range("P11:P" & numIterations + 10) ' Only plot errors
    chart.ChartType = xlLine
    chart.HasTitle = True
    chart.ChartTitle.Text = "Learning Curve"
    chart.Axes(xlCategory).HasTitle = True
    chart.Axes(xlCategory).AxisTitle.Text = "Iteration" ' Add title to x-axis
    chart.Axes(xlCategory).CategoryNames = sheet.Range("O11:O" & numIterations + 10) ' Set
x-axis labels to iterations
    chart.Axes(xlValue).HasTitle = True
    chart.Axes(xlValue).AxisTitle.Text = "Error"

    ' Notify user that the process is complete
    MsgBox "Process complete. Review the results, then run 'ResetRandomValues' to reset the
weights to random values."
End Sub

Sub ResetRandomValues()
    Dim sheet As Worksheet
    Set sheet = ThisWorkbook.ActiveSheet
```

```
    ' Reset the cells with RAND()
    sheet.Range("F2:F4").Formula = "=RAND()"
    sheet.Range("G2:G4").Formula = "=RAND()"
    sheet.Range("H2:H4").Formula = "=RAND()"
    sheet.Range("I2:I4").Formula = "=RAND()"

    MsgBox "Values have been reset to RAND()."
End Sub
```