John Jakobsen

Numerical Issues in Scientific Programming

December 16th, 2022

Math Behind Neural Networks

Introduction

A Neural Network is a collection of layers, each with a collection of neurons in them. Each neuron connects to some neurons in the next layer, with each connection having a strength or weight to it. They are modeled after actual neurons in our brains (a very simple understanding of them).

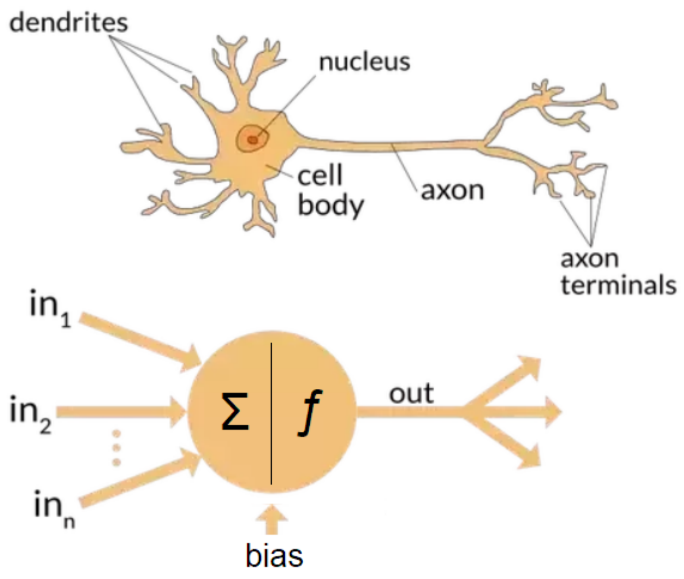


Fig 1. A Modeled Artificial Neuron (Pictured Below) and an Actual Neuron (Pictured Above)

In Fig. 1, the top model represents an actual neuron; it contains inputs to the neuron, the dendrites, which take in electrical signals. That input then gets processed in the neuron then outputs their own signals through the axon terminals. In our brains we have an average of 86 billion neurons each connecting with other neurons to relay information throughout our brains.

The bottom model in Fig.1 represents an artificial neuron. Similarly to the neuron, the artificial neuron (A.N.) takes in a number of inputs then assigns a weight to that input. Each weight is multiplied by it's corresponding input. Then they are added up and this sum is put through an activation function. The output of the activation function is the output for the A.N.
This process is called forward propagation.

Coming up with the answer provided a given set of inputs is easy enough, you just need to go through this process for each A.N. then propagate the outputs of every cell in each layer to the inputs of the cells in the next layer until you reach the output layer. However, actually teaching the network is different. This process is called back propagation. In back propagation we take the neural networks output for a set of inputs, then we check how "wrong" the network was, we then update the weights based on how wrong the network was. In this process we are propagating the error backwards through the neural network from the output layer to the input layer.

Forward Propagation

In this section we will go deeper into how forward propagation works in a Neural Network. The Pseudo Algorithm for how forward propagation works is as such

```
For each layer, L:
    For each node, n, in L:
        net = 0
        For each input, i,  to n:
            net += $w_i * i$
        n.output = Activation(net + bias)
```

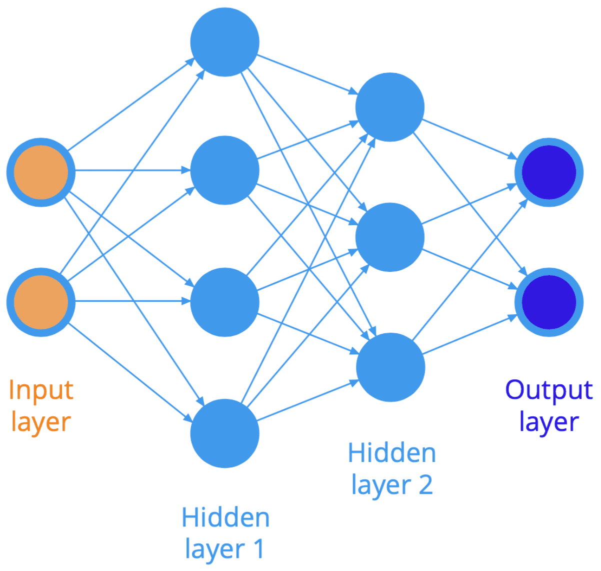


Fig. 2 An Artificial Neural Network with 2 hidden layers, 2 inputs and 2 outputs.

In the above figure, we'd first calculate the outputs of the input layer for each node, however this is just the inputs. Then for every input into every node in Hidden Layer 1 we multiply the weights by the inputs (In the figure this is represented by the lines between Input Layer and Hidden layer 1. Proceed to do this for Hidden Layer 1 to Hidden Layer 2, then from Hidden Layer 2 to Output Layer.

Example

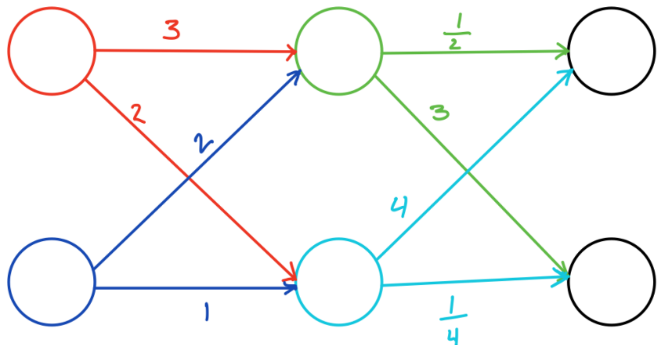


Fig. 3 Example Artificial Neural Network with 1 hidden layer, 2 inputs and 2 outputs. The numbers correspond to each connection's weight and the colors correspond to the node the output belongs to.

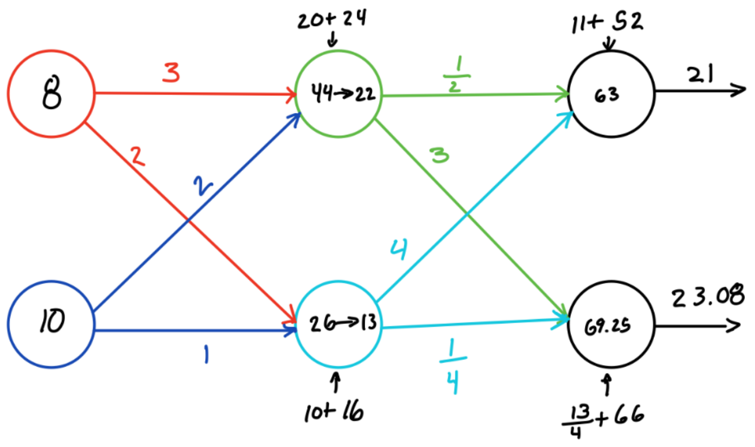
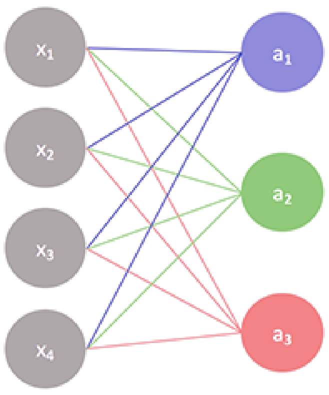


Fig 4. Example Artificial Neural Network from Fig. 3 with the outputs of each neuron calculated (denoted by the black numbers in each circle, the nodes)

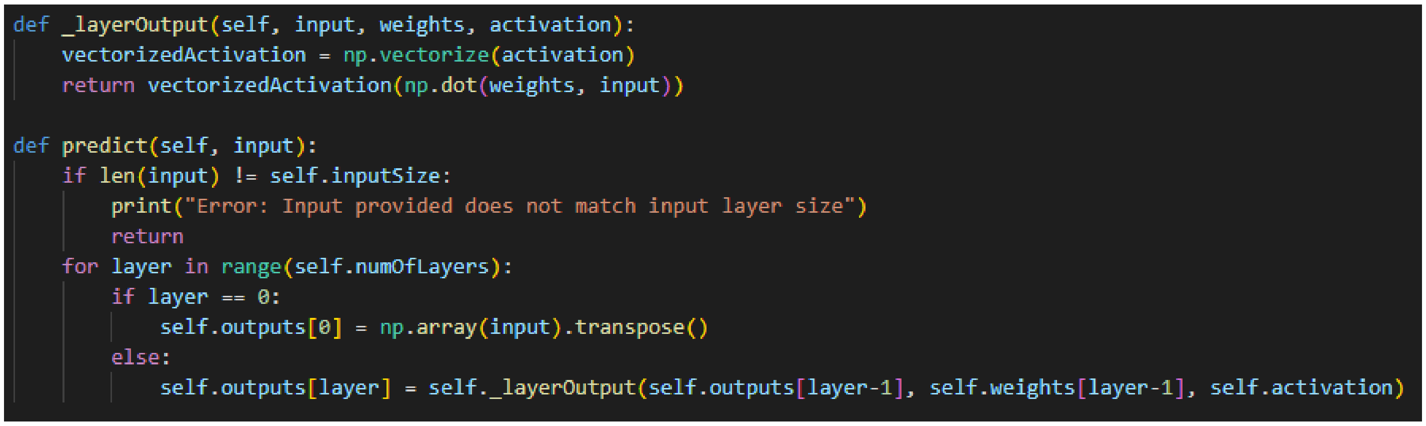
To represent this in Linear Algebra we can store the weights between each layer as an m x n matrix (m nodes in the right layer and n nodes in the left layer), with the rows representing the weights from the left layer to the right layer. We can denote each weight as w_{ij}, where i represents the node in the right layer the weight connects to and j represents the node in the left layer the weight connects to. We'll call this matrix W. The inputs we can represent as an m x 1 vector, X. The Biases we can represent as an m x 1 matrix, B. So to get the output for each layer we do $\sigma(W \bullet X + B)$. Where σ is the activation function.



$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Fig. 5 Example of a neural network layer output being computed via matrices

Code



```python
def _layerOutput(self, input, weights, activation):
    vectorizedActivation = np.vectorize(activation)
    return vectorizedActivation(np.dot(weights, input))

def predict(self, input):
    if len(input) != self.inputSize:
        print("Error: Input provided does not match input layer size")
        return
    for layer in range(self.numOfLayers):
        if layer == 0:
            self.outputs[0] = np.array(input).transpose()
        else:
            self.outputs[layer] = self._layerOutput(self.outputs[layer-1], self.weights[layer-1], self.activation)
```

The _layerOutput function computes the output of a layer given the input vector, weight matrix and activation function. We then apply the function to each layer and save the outputs for the next layer to be used as inputs.

Backward Propagation

In this section we will go deeper into how backward propagation ("Back Propagation") works. The Pseudo Algorithm is as such

```
Train the network on one example using forward propagation
For each Layer (Starting from the layer behind the output layer and moving back to the input layer), L:
    For each node, N, in L:
        For each weight in node's weights, W:
            weightChangeMatrix[L][N][W] = InputForNode(L, N) * outputForNode(L,
```

*N) * (1 - outputForNode(N)) * affectOnOutputOf(N) / NumberOfSamplesTraining*

Do this for all the samples training then once we are done, for each layer matrix, we add weightChangeMatrix[L] to it.

We start from the weight matrix between the output layer (Layer L) and the layer behind it (Layer L - 1). For each weight we must find the derivative of the total error with respect to the weight. We can use the chain rule to accomplish this since the error is just a long composed function with each weight.

$$\frac{\delta Error_{total}}{\delta w_{ij}} = \frac{\delta Error_{total}}{\delta out_j} * \frac{\delta out_j}{\delta net_j} * \frac{\delta net_j}{\delta w_{ij}}$$

To calculate this equation we first need to define an error function and this function must be differentiable. We will choose the squared error defined below:

$$Error_{total} = \frac{1}{2}(target_j - out_j)^2$$

With $target_j$ being the actual value provided for the sample for $node_j$ and out_j being the output the network calculated during forward propagation for $node_j$.

Notice also that we have not defined an Activation function yet, we will do this here. Our activation function must also be differentiable. For this we will use the sigmoid function, this function accepts all real values and outputs a real value from 0 to 1.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Now that we have the output function and error function defined we can start calculating the change in total error with respect to the weight.

$$\frac{\delta Error_{total}}{\delta out_j} = -(target_j - out_j)$$

$$\frac{\delta out_j}{\delta net_j} = out_j(1 - out_j)$$

$$\frac{\delta net_j}{\delta w_{ij}} = y_i$$

Here y_i is the ith input (corresponding to w_{ij}).

In order to calculate $\frac{\delta Error_{total}}{\delta out_j}$ for the next layer we also need to calculate the change in error with respect to the change in the layer's input (or the the previous layer's output)

$$\frac{\delta net_j}{\delta y_i} = w_i$$

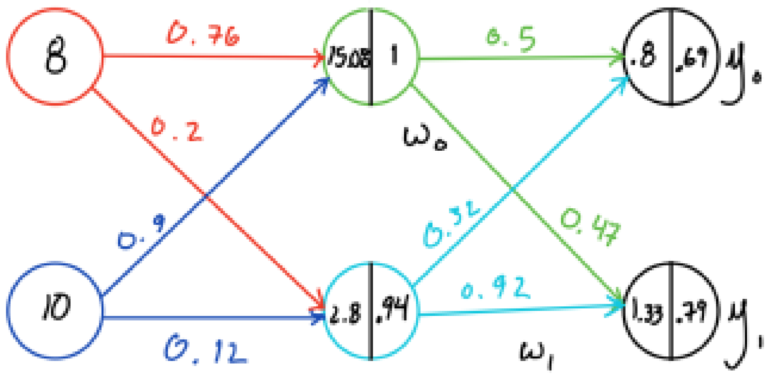
So to find the change in error with respect to the output in the previous layer we need to add up all the change in error for each node that accepts that input.

$$\frac{\delta Error_{total}}{\delta y_i} = \sum_j \left(\frac{\delta Error_{total}}{\delta out_j} * \frac{\delta out_j}{\delta net_j} * \frac{\delta net_j}{\delta y_{ij}} \right)$$

This gives us how $node_j$ in the previous layer affects the error (Since it is connected to every node in the current layer).

Applying this to every node's output in the previous layer we can save these results and supply it to the next calculations on the previous layer, to use as the change in error with respect to each node's output.

Example



Using the Neural Network that was used in the Forward Propagation section, we will work out the change to make to one weight (w_1) in the example above.

Let the actual value of $y_1 = 1$.

In the example, out_1 of the output layer is 0.79.

$$\frac{\delta Error_{total}}{\delta out_j} = -(target_j - out_j) = -(1 - 0.79) = -0.21$$

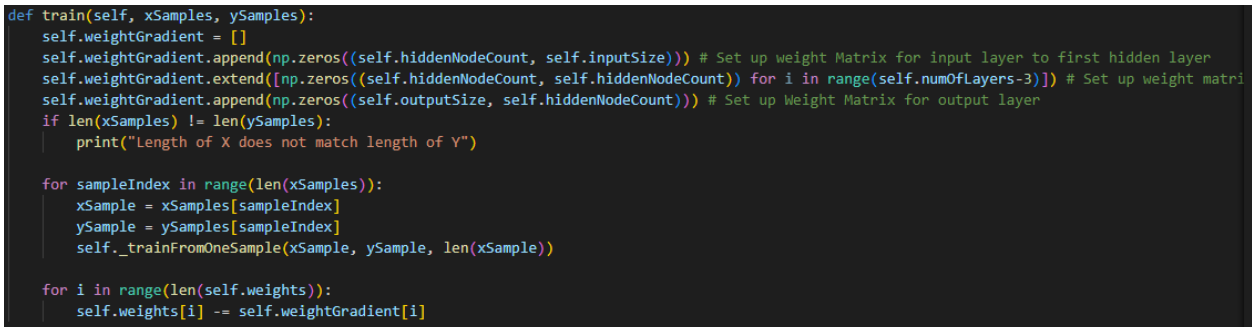
$$\frac{\delta out_j}{\delta net_j} = out_j(1 - out_j) = 0.79(1 - 0.79) = 0.1659$$

$$\frac{\delta net_j}{\delta w_{ij}} = 0.92$$

$$\frac{\delta Error_{total}}{\delta w_{ij}} = \frac{\delta Error_{total}}{\delta out_j} * \frac{\delta out_j}{\delta net_j} * \frac{\delta net_j}{\delta w_{ij}} = (-0.21)*(0.1659)*(0.92) = -0.03205188$$

So we would use this value (-.03205188) to subtract from w_1. Which would nudge w_1 to 0.95205188.

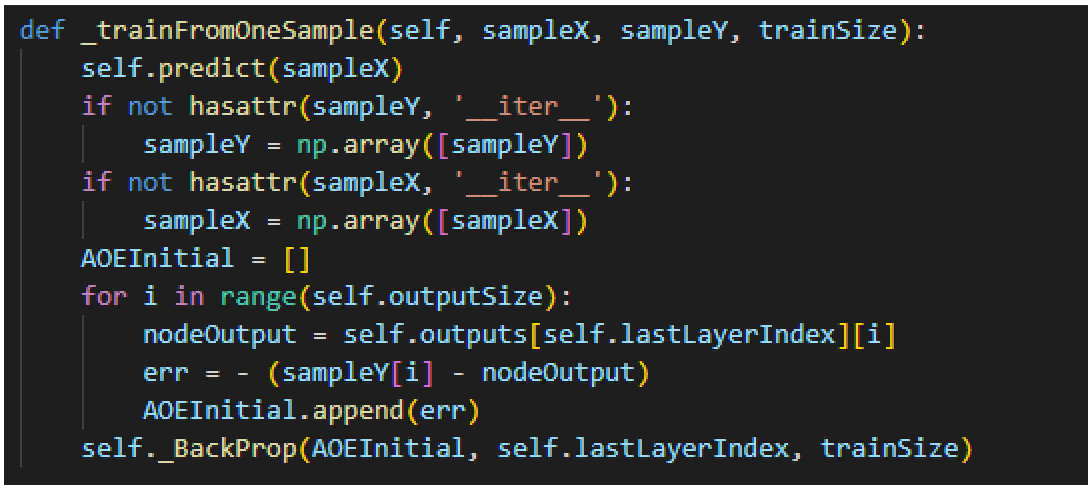
Code



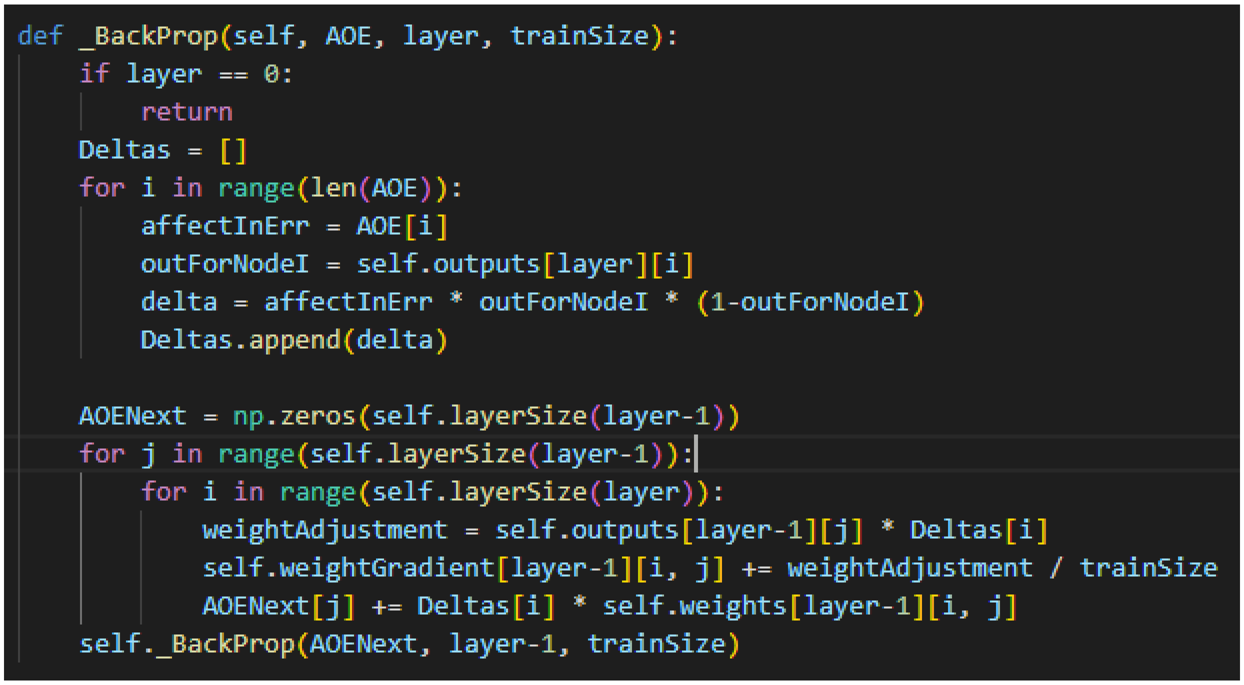
```
def train(self, xSamples, ySamples):
    self.weightGradient = []
    self.weightGradient.append(np.zeros((self.hiddenNodeCount, self.inputSize))) # Set up weight Matrix for input layer to first hidden layer
    self.weightGradient.extend([np.zeros((self.hiddenNodeCount, self.hiddenNodeCount)) for i in range(self.numOfLayers-3)]) # Set up weight matri
    self.weightGradient.append(np.zeros((self.outputSize, self.hiddenNodeCount))) # Set up Weight Matrix for output layer
    if len(xSamples) != len(ySamples):
        print("Length of X does not match length of Y")

    for sampleIndex in range(len(xSamples)):
        xSample = xSamples[sampleIndex]
        ySample = ySamples[sampleIndex]
        self._trainFromOneSample(xSample, ySample, len(xSample))

    for i in range(len(self.weights)):
        self.weights[i] -= self.weightGradient[i]
```



```
def _trainFromOneSample(self, sampleX, sampleY, trainSize):
    self.predict(sampleX)
    if not hasattr(sampleY, '__iter__'):
        sampleY = np.array([sampleY])
    if not hasattr(sampleX, '__iter__'):
        sampleX = np.array([sampleX])
    AOEInitial = []
    for i in range(self.outputSize):
        nodeOutput = self.outputs[self.lastLayerIndex][i]
        err = - (sampleY[i] - nodeOutput)
        AOEInitial.append(err)
    self._BackProp(AOEInitial, self.lastLayerIndex, trainSize)
```



```
def _BackProp(self, AOE, layer, trainSize):
    if layer == 0:
        return
    Deltas = []
    for i in range(len(AOE)):
        affectInErr = AOE[i]
        outForNodeI = self.outputs[layer][i]
        delta = affectInErr * outForNodeI * (1-outForNodeI)
        Deltas.append(delta)

    AOENext = np.zeros(self.layerSize(layer-1))
    for j in range(self.layerSize(layer-1)):
        for i in range(self.layerSize(layer)):
            weightAdjustment = self.outputs[layer-1][j] * Deltas[i]
            self.weightGradient[layer-1][i, j] += weightAdjustment / trainSize
            AOENext[j] += Deltas[i] * self.weights[layer-1][i, j]
    self._BackProp(AOENext, layer-1, trainSize)
```

Sources

Brain Basics: The Life and Death of a Neuron | National Institute of Neurological Disorders and Stroke. https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-life-and-death-neuron#:~:text=Neurons%20are%20information%20messengers.,rest%20of%20the%20nervous%20system. Accessed 11 Dec. 2022.

Herculano-Houzel, Suzana. "The Remarkable, yet Not Extraordinary, Human Brain as a Scaled-up Primate Brain and Its Associated Cost." Proceedings of the National Academy of Sciences, vol. 109, no. supplement_1, June 2012, pp. 10661–68. DOI.org (Crossref), https://doi.org/10.1073/pnas.1201895109.
Accessed 11 Dec. 2022.

Nagyfi, Richard. "The Differences between Artificial and Biological Neural Networks." Medium, 4 Sept. 2018, https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7.

"What Is the Differences between Artificial Neural Network (Computer Science) and Biological Neural Network?" Quora, https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network. Accessed 11 Dec. 2022.

Mazur. "A Step by Step Backpropagation Example." Matt Mazur, 17 Mar. 2015, https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/.

But What Is a Neural Network? | Chapter 1, Deep Learning. www.youtube.com, https://www.youtube.com/watch?v=aircAruvnKk. Accessed 11 Dec. 2022.

Gradient Descent, How Neural Networks Learn | Chapter 2, Deep Learning. www.youtube.com, https://www.youtube.com/watch?v=IHZwWFHWa-w. Accessed 11 Dec. 2022.

What Is Backpropagation Really Doing? | Chapter 3, Deep Learning. www.youtube.com, https://www.youtube.com/watch?v=Ilg3gGewQ5U. Accessed 11 Dec. 2022.