Quick Introduction to Neural Networks

By Max Gudwin, ECE '21

Introduction

"Machine learning" is a term that gets thrown around quite a bit in the fields of electrical and computer engineering. Despite this, many engineers have not yet had to work with machine learning. The result is a knowledge gap on how machine learning works. The team Maximum Red Senior Design project used an Artificial Neural Network (ANN) with supervised learning to classify EMG signals. This tech note is dedicated to demystifying this part of the project by giving a light introduction on the basics of Neural Networks with supervised feed-forward learning. Specifically, this tech note walks though the parts of a feed forward neural network with supervised learning. Because neural networks are the subject of incredible amounts of research, this tech note will not go into detail on every subject, but it can hopefully be used as a strong starting point for further exploration.

Anatomy of a neural network

To understand what a neural network is, a good place to start would be looking at a diagram (Figure 1). A diagram of a neural network usually looks like a bunch of circles in columns or rows connected together with lines. In such a diagram, columns can be organized into three categories: the input layer, the hidden later, and the output layer. In a *feed-forward* configuration, information starts at the input later and moved towards the output layer. The goal is that data will come in though the input, and a decision about what the data means will leave through the output.

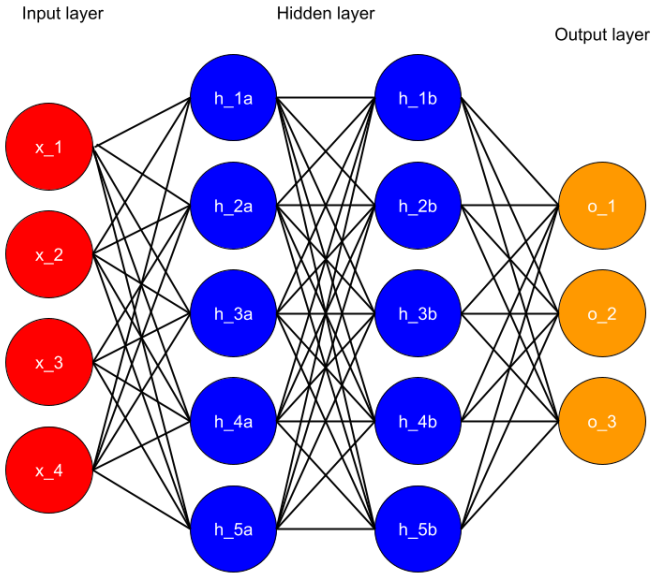

Figure 1: Diagram of a generic neural network

Input Layer

The input layer is simply made of the numbers being passed into the system. Every input number is represented by a circle. For example, if a neural network were built to recognize, and categorize apples and oranges in an image each input in the input layer could be an R, G, or B pixel value of a pixel within the image. In this setup, the network diagram would need as many input circles as there are pixels in the image (Figure 2).

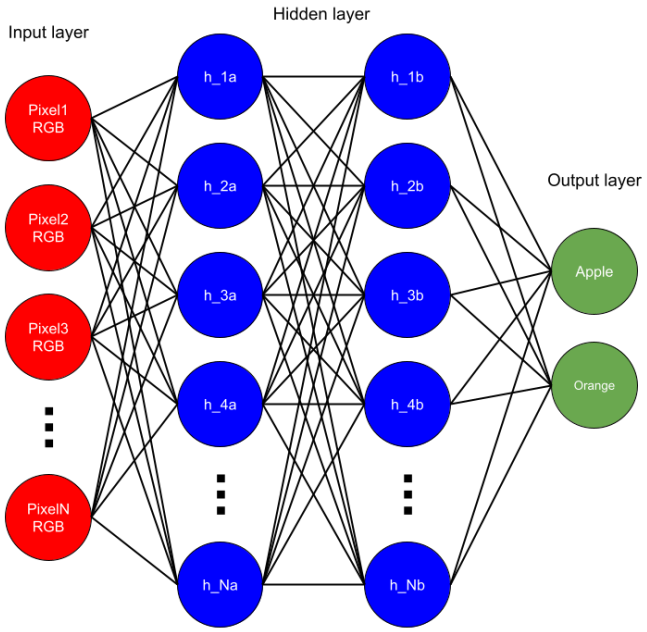

Figure 2: Neural network that finds apples and oranges

It is important to note that in practice creating a neuron for each pixel in an image recognition problem is usually too complicated. Instead, some sort of *feature extraction* is used to find significant information and give it to the neural network. In the example of apple vs orange classification, one useful feature to extract could be how much orange, or how much red is in the picture. That, along with some other features, will be far simpler to process than pixel values.

Hidden Layers

Hidden layers are the layers between the input and output layers and the hidden layers consist of every circle between the input and output. In a hidden layer, each bubble is a neuron and the connections of each neuron are represented by the lines connecting each neuron to its neighbors. The neuron's connections closer to the input layer are considered the inputs for that neuron, and the ones on the output sides are the output connections. In a fully connected layer, neurons pass their information to all the neurons in the next layer. Fully connected networks tend to be computationally expensive and many workarounds exist such as convolutional neural networks.

Hidden layers have many parameters about them that can be tweaked compared to the input and output layers. This includes the number of neurons per layer and the number of hidden layers. The number of neurons required for a layer is determined by the complexity of a problem. One general rule of thumb is that, supposing that inputs and outputs are based on two features, the number of intersecting lines needed to separate the inputs and outputs on a graph correspond to the minimum number of neurons needed in a hidden layer. By this rule, if the input and output data is linearly separable only an input and output layer are needed. Generally, classification problems don't use more than two hidden layers.

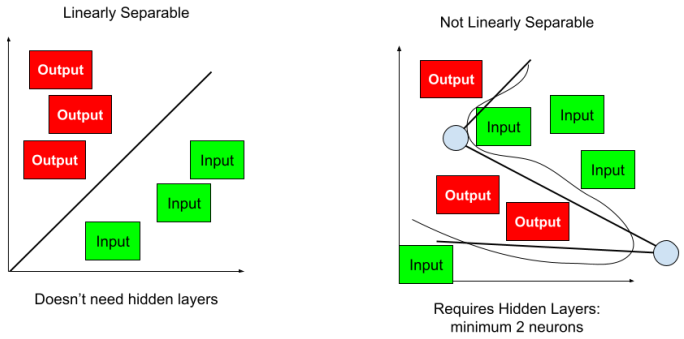

Figure 3: How many hidden neurons are needed?

Output Layer

The output layer, like the hidden layers, is also made of neurons. The only difference is that the outputs of this layer of neurons are the network's solution to the problem. No line is drawn from the neuron bubbles of the output layer to anywhere else because the pathway ends here. Finding the number of neurons in this layer is simple: conventionally it is just the number of answers the problem can have. In the case of a network that finds apples and oranges the output could be two neurons: one apple neuron and one orange neuron. The apple and orange neurons can each report a confidence value of whether an apple or orange is found. Another option could be each output neuron giving a one or zero based on whether the fruit is present. How the outputs formatted depends entirely on what it will be used for and is up to the designer. For example, network that "reads"

letters of the alphabet would probably have 26 output neurons.

Anatomy of a neuron

The next step in understanding how a neural network works is to understand how an individual neuron works. In a neuron, all information is processed the same way.

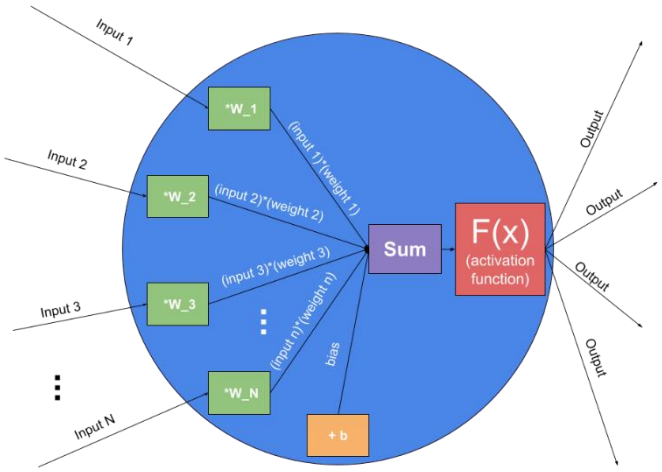

Figure 4: Anatomy of a neuron

Weights

First, every input to the neuron is multiplied by a weight, or weighted. Weights for each input are different and correspond to what the neuron believes is the relationship between this input and the desired output. For example, if the input had absolutely no correlation with the output, the weight for that input might fall to be 0 or near 0. Next, the neuron takes the sum of all the inputs as well as the bias in order to combine them so they can be placed in the activation function.

Activation Function

The result is then given to the activation function. The activation function determines how much the output of the neuron should be turned on or off given all the weighted inputs. For example, the last activation function of a network finding apples and oranges will determine whether, given all the information from the previous layer, the answer is an apple or an orange.

Mathematically, the activation function makes the neural network non-linear. Without an activation function, it would be possible to model any neural network with a single layer of neurons because the answer would be a linear transformation of the input. This only works if a clean line can be drawn separating inputs and outputs. For this reason, less linear the problem is the more neurons are required in the hidden layers.

There are many choices for what can be used as an activation function such as ReLu, Softmax, and arctan, each offering their own advantages. All activation functions follow the same rules: they must be non-linear and differentiable. A popular function is the sigmoid function:

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

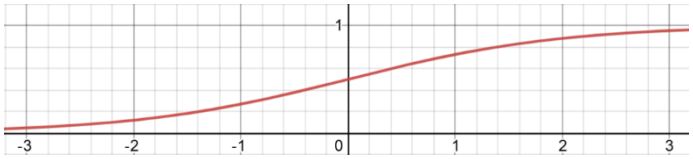

Figure 5: Sigmoid function

The sigmoid function can take any real number and converts it to a value between 1 and 0. Its advantage is its strong roots in stochastic systems. Its disadvantage is its shallow slope at its extremes which affect learning speed.

Finally, with the data through the activation function, the result is given to each neuron the in the next layer, or, if it is the output layer the result is the output.

Training

Neural networks are a form of machine learning, and for a machine to learn it must be taught. This tech note will only discuss supervised learning as it is relevant to the Team Maximum Red senior design project.

To begin supervised learning we must start with labeled data. In the case of apples and oranges we must have many images of apples and oranges and

already know which one is an apple and which one is an orange. This is called our training set.

Backpropagation

Learning works by adjusting the weights of each neuron as a function of the error, cost, or loss found in the output of a neural network vs the desired output. The process of learning in a neural network is called backpropagation. It is called this because error is found and propagated backwards through each layer of the network while weights are adjusted. The idea is to adjust the weights based on how much they affect the error. To find out how the weights affect the error we need:

1. The partial derivative of error with respect to the output of the neuron

$$\frac{\partial E}{\partial \text{Out}}$$

2. The partial derivative of output with respect to the input of the neuron

$$\frac{\partial \text{Out}}{\partial \text{In}}$$

3. The partial derivative of the input with respect to the weights of the neuron

$$\frac{\partial \text{In}}{\partial W}$$

The error can be found using an error function such as cross entropy or means squared error.

$$E = \frac{-1}{n}\sum_{i=1}^{n}\left(a_i \times ln(\text{Out}_i) + \left((1-a_i) \times ln(1-\text{Out}_i)\right)\right)$$

Equation 1: cross entropy formula n = # of neurons, a = right answer, Out = neuron's answer

Then, calculate the derivative of the error formula and activation function used in order to use it in the weight adjustment formula.

$$\frac{\partial E_i}{\partial \text{Out}_i} = \frac{-\partial\left(a_i ln(\text{Out}_i) + (1-a_i) ln(1-\text{Out}_i)\right)}{\partial \text{Out}_i}$$

$$\frac{\partial E_i}{\partial \text{Out}_i} = -\left(a_i\left(\frac{1}{\text{Out}_i}\right) + (1-a_i)\left(\frac{1}{1-\text{Out}_i}\right)\right)$$

Equation 2: Partial derivative of the cross-entropy formula for a single output

Combining all three we can get the partial derivative of the error with respect to the weight of an input on the neuron. In essence we have the "slope" of the error vs weight function in and we know in which direction to step in order to reduce the error.

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \text{Out}}\frac{\partial \text{Out}}{\partial \text{In}}\frac{\partial \text{In}}{\partial W}$$

Equation 3: Slope of Error over Weight

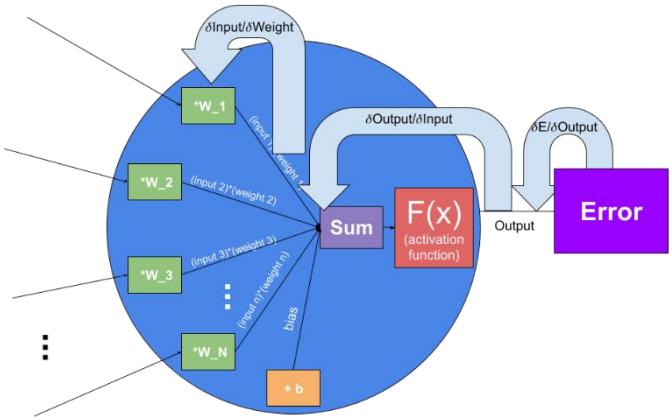

Figure 6: Backpropagation on a single neuron

Gradient Descent

So, we now reduce each weight by the slope times some "learning rate." The "learning rate" is essentially the size of each step taken. Because we are going down the slope of the error function to get less error these methods are referred to as "gradient descent." Some more advanced gradient descent algorithms include terms such has "momentum" which can change the size of the steps taken based on previous steps. However, a very simple weight adjustment formula would look like this:

$$W_{new_i} = W_{old_i} - (\text{learning rate})\frac{\partial E_{total}}{\partial W_{old_i}}$$

Equation 4: Simple gradient descent equation

This process must be repeated for every weight of every neuron in every layer. For neurons connecting to multiple neurons on their output side, the partial derivative of error with respect to each one of the outputs must be found in order to get a partial derivative of total error vs output. So, the error of the last layer must be fully computed before backpropagation i the next layer can begin.

Because backpropagation is computationally expensive, error can be accumulated over more than one training example before weights are updated. Giving the whole training set is given before updating weights is called “batch” gradient descent. Giving only a small part of the set is called “mini batch” gradient descent. And giving only one example is called “stochastic” gradient descent because its direction will be more random.

As a result of all these calculations. the most computationally expensive part of making a neural network is usually training it. But there is more, this whole process only adjusted the weights by one step in the right direction. Much like a student studying for an exam or memorizing flashcards we must repeat this training and adjustment many time over a complete training data set after all, you probably wouldn’t be good at something after only practicing it once. The number of times the network goes through the whole data set is called the epoch.

Conclusion

Though there are many actual design choices for a classifying feed forward neural network using supervised learning that make ANNs different from one another they all follow these same general rules and structures. Hopefully, this gives some insight as to how the computer is “thinking” in the neural prosthetic project.

To take another look at some of the topics glossed over there are links to some free online articles in the References section. I would recommend [8] for convolutional methods, [10] for activation functions, [6] for learning rate, [9] to get started coding on your own, and [5] for a full mathematical backpropagation walkthrough.

References

1. Yadav, Neha, Yadav, Anupam, and Kumar, Manoj. An Introduction to Neural Network Methods for Differential Equations. SpringerBriefs in Applied Sciences and Technology. Computational Intelligence. 2015.

2. Priddy, Kevin L, Keller, Paul E, and Society of Photo-optical Instrumentation Engineers. Artificial Neural Networks : An Introduction. Tutorial Texts in Optical Engineering ; TT68. Bellingham, Wash.: SPIE, 2005.

4. Braspenning, P. J., Thuijsman, F., and Weijters, A. J. M. M. Artificial Neural Networks : An Introduction to ANN Theory and Practice. Lecture Notes in Computer Science ; 931. Berlin ; New York: Springer, 1995.

5. Prakash Jay. "Backpropagation is very simple. Who made it Complicated?" Medium. April 20th 2017. https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

6. Brownlee Jason. “Understand the Impact of Learning Rate on Neural Network Performance.” Machine Learning Mastery. January 25 2019. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

7. Yadav Saurabh. “Weight Initialization Techniques in Neural Networks.” Towards Data Science. November 9th 2018. https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

8. Hue Antoine. “Dense or Convolutional Neural Network.” Medium. January 28 2020. https://medium.com/analytics-vidhya/dense-or-convolutional-part-1-c75c59c5b4ad

9. Spencer-Harper Milo. “How to build a simple neural network in 9 lines of Python code.” Medium. July 21 2015. https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1

10. Gharat Snehal. “What, Why and Which?? Activation Functions.” Medium. April 14th 2019. https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441

11. Sarang, P. G. Artificial Neural Networks with TensorFlow 2 : ANN Architecture Machine Learning Projects. S.l.]: Apress, 2021.