Neural Networks

Gautom Das, Liam DeVoe, Nate Kelkay

Introduction

Before we begin, we would like to briefly go over words that are commonly thrown around but **may not** be used interchangeably in the discussion of neural networks. As we have already discussed artificial intelligence and machine learning —any algorithm with the ability to learn— we will start by defining deep learning: a subset of machine learning that take advantage of neural networks. The different fields relate to one another as subfields as seen below:

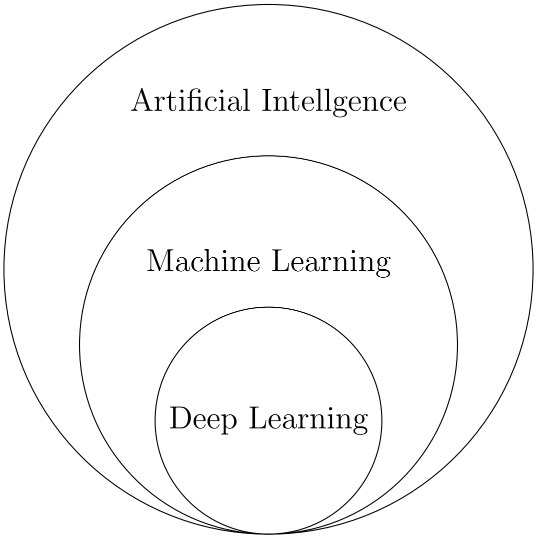


Figure 1: Visual of the subfields.

Field

Let's start with a single neuron from the brain. Dendrites feed information into it, then there is the body of the neuron that performs some 'biology' operation on the inputs then outputs this information with its axons.

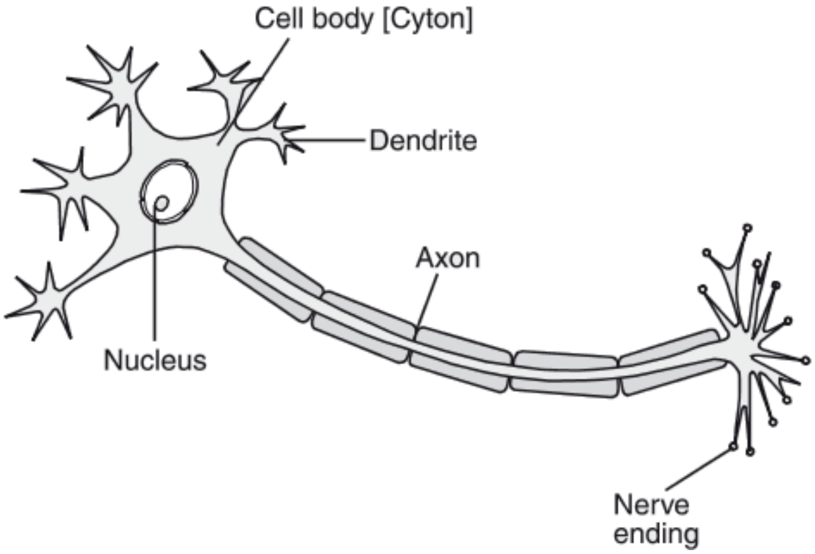
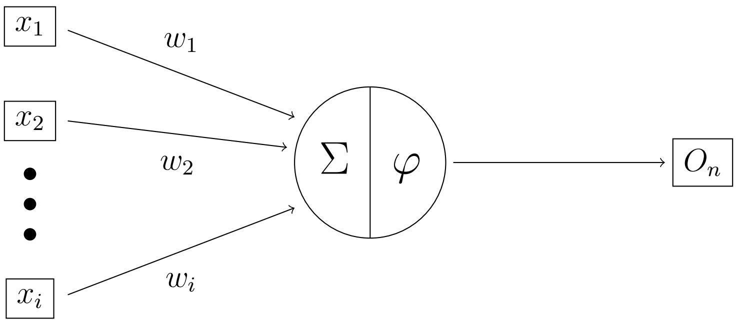


Figure 2: Diagram of a dendrite

With this understanding, the following model for a neuron may be developed.

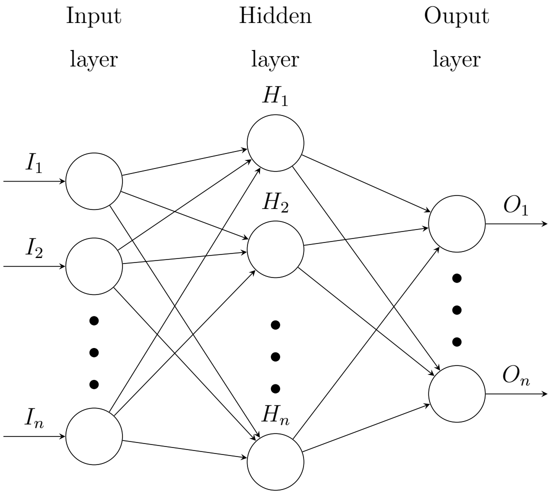


Similar to the biological description for a neuron, this model feeds in information denoted x_n, multiplies each input by its own weight denoted w_n then feeds into some function that sums all the information. From there an additional function may be performed on the input and then this value is passed onto the next set of neurons. The formal formula for this neuron also takes advantage of a bias:

$$O_n = \varphi\Big(\sum_{i=1}^{n} x_i w_i + b\Big)$$

This function φ is the **activation function**. An activation is any function that takes some real number and maps it to a constrained function. Common examples are the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$ and the tanh function. Intuitively, the reason an activation function is needed is because the inputs and weights can theoretically become any number. Without an activation function, very large inputs/weights would dominate the network and skew learning.

Combining many of these networks gives you a perceptron as seen below:



Forward Propagation

A perceptron is our neural network. Inputting the values at the start of the input layer and then running it through the network will give you the predicted output. To derive the general formula for a vector describing the weights at any given hidden layer. Starting with the following formula for a given value in the hidden layer denoted H_n:

$$H_n = w_1 i_1 + w_2 i_2 + ... + w_n i_n$$

A keen observer may recognize the implementation of matrix multiplication. A general formula for a given hidden layer is given below:

$$H = x^T w + b$$

Cost Function

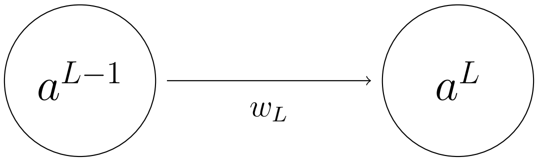
Now we have two vectors: the output layer of the model and the expected value. In order to train our model we need to start by evaluating our error. The method that is most commonly used in regression functions is Mean-Squared Error (MSE):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - \hat{y})^2$$

For different applications there are specific functions that perform better than others. The cost function itself is the average of all the different costs in a training data set.

Backward Propagation

Now we finally have our error how do we actually edit the neural network? We take advantage of an algorithm known as backpropagation. To best understand this, let's set up a very basic neural network with two neurons:



Let's start by calculation the cost of our last node a^L versus the real output y:

$$C = (a^L - y)^2$$

From here, we want to figure out how sensitive the cost function is to the weight w^L. In more formal terms we want $\frac{\partial C}{\partial w^L}$. Knowing how each value in a neural network is connected we want to first define a relationship between a^{L-1} and a^L:

$$z^L = w^L a^{L-1} + b^{L-1}$$

$$a^L = \varphi(x^L)$$

Now we can establish a clear relationship between the cost and the weight:

$$\frac{\partial C}{\partial w_L} = \frac{\partial z^L}{\partial w_L} \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L}$$

Doing the specific partial derivatives we can solve each value in our chain rule to get our final value:

$$\frac{\partial z^L}{\partial w_i} = a^{L-1}$$

$$\frac{\partial a^L}{\partial z^L} = \varphi'(z^L)$$

$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

We have now our final cost formula of:

$$\frac{\partial C}{\partial w^L} = a^{L-1} \varphi'(z^L) 2(a^L - y)$$

Our gradient for the function is the average of all of these cost functions in relation to every output and weight/bias:

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \frac{\partial C}{\partial b^1} \\ \vdots \\ \frac{\partial C}{\partial w^L} \\ \frac{\partial C}{\partial b^L} \end{bmatrix}$$

Finally, we have the gradient descent algorithm. Gradient descent is an iterative algorithm which takes steps to reduce a function. Thus we first have to define a learning rate we call α. We also need our cost for the current input. From there given a certain point Θ_j we can define the best path to reduce it's weight as:

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} * C$$

This just is our gradient descent algorithm. Formally you would combine this with the cost function, but as of now you have gained the building blocks behind neural networks.

Convolutional Neural Networks

For convolutional neural networks, the goal of the 'convolutions' is to convert the images to an array of features. To do this, a convolutional neural network takes advantage

References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks *University of Toronto*, 2012.

[2] Ian Goodfellow, Yoshua Bengio, Aaron C. Courville, Deep Learning *MIT Press*, 2015.

[3] Peter Norvig, Stuart J. Russell, Artificial Intelligence: A Modern Approach *Prentice Hall*, 1994.

[4] Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning. *Prentice Hall*, 2001.