Paper Title

Abstract

Our abstract.

1 Introduction

Introduction here.

2 Neural Network Background

The human brain is composed of billions of neurons: interconnected electrically excitable cells that form the basis of our intelligence. Each neuron has synapses that receive electrical signals from multiple other neurons. If the sum of these inputs is greater than a certain value, the neuron fires, generating a voltage at its axon. The axon, or the output of the neuron cell, is itself connected to a synapse of another neuron. The interconnections of these neurons form a massive network, where a huge number of inputs are processed in parallel to a set of outputs. The basis of artificial neural networks is to simulate the mathematical properties of these neurons in order to perform similar tasks of mass parallel computation.

2.1 Single Artificial Neuron

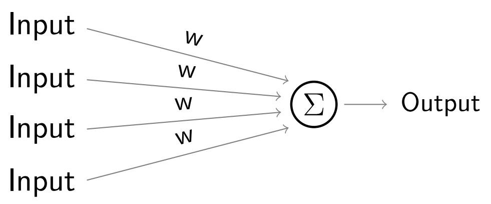

Figure 1: Single Neuron Diagram

An artificial neuron simply a mathematical model of a biological neuron, and therefore its functionality is very similar. Each neuron has multiple inputs, each with an individual weight, and one output. The weight of an input is simply a positive or negative fraction that govern the impact of each input on the single output. An artificial neuron's set of weights determines its function, and the function of each neuron in a neural network (in addition to the arrangement of the network) determines the function of the network as a whole. This will become important as we discuss training Fido's neural network, however for the moment we will adjust our focus to the output of an artificial neural network. By summing each input multiplied by its individual weight we reach a value called the activation, expressed as such mathematically for each input x and weight w:

$$a = \sum_{i=0}^{i=n} x_i w_i \,. \tag{1}$$

If this activation value is less than a certain threshold, the output is zero. If it is greater than the threshold, the output is one. This activation function most closely resembles the biological model of a neuron, a binary step function. However, a binary output can be somewhat limiting for many applications of neural networks. For example, many of Fido's outputs are gradient rather than linear; as an example, an LED can be given a brightness. For this purpose alternate activation functions can be used with gradient outputs. One such function is a sigmoid function, expressed as such:

$$O(a) = \frac{1}{1 + e^{-\frac{a}{p}}} \,, \tag{2}$$

for each output O, activation a, and constant p. The sigmoid activation function can also be graphed as below.

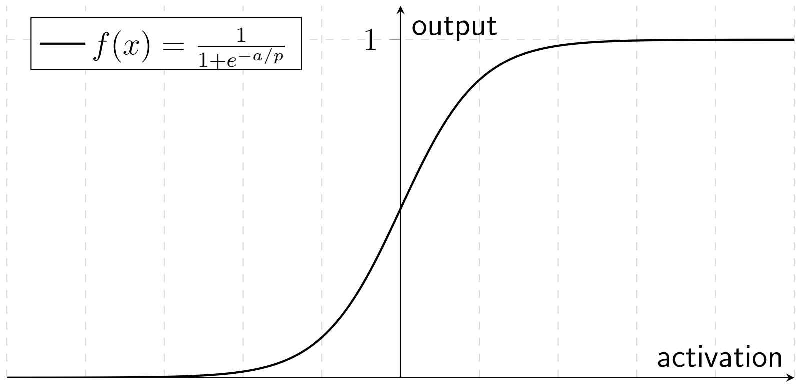

Figure 2: Sigmoid Function Graph

This provides us with a gradient output, however output is still limited to positive values. An alternative activation function which allows outputs ranging from -1 to +1 is the hyperbolic tangent activation function:

$$O(a) = \tanh(a)\,. \tag{3}$$

The hyperbolic tangent activation can be graphed as such, demonstrating its greater range and gradient output.

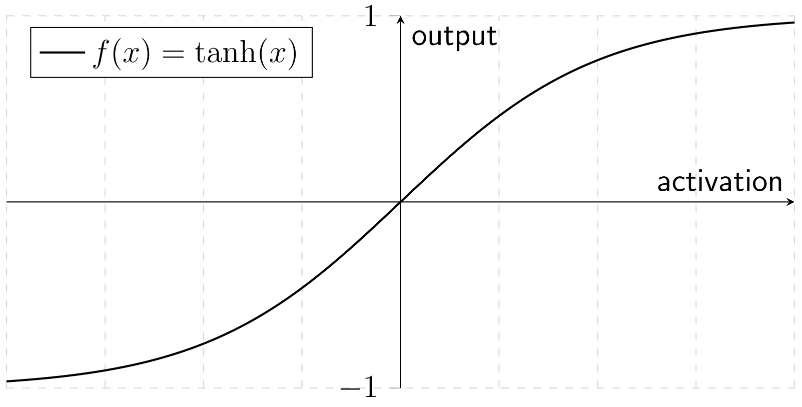

Figure 3: Hyperbolic Tangent Function Graph

2.2 Feedforward Neural Network

A traditional method of arranging artificial neurons in a neural network is called a feedforward network. Neurons are connected as previously described: the output of each neuron is connected to one of the inputs of another neuron. These neurons are organized into layers, as described in the following figure:

Initial inputs are first sent into the input layer. Next, the output of each input layer neuron is sent into each neuron in the hidden layer. The hidden layer processes the inputs to outputs using an activation function and a weight value for each input. There can be any number of hidden layers in a neural network, depending on the complexity of the computation being performed. Finally the outputs from the last hidden layer neurons are sent into each neuron of the output layer, where the final outputs are processed. A concrete example of a

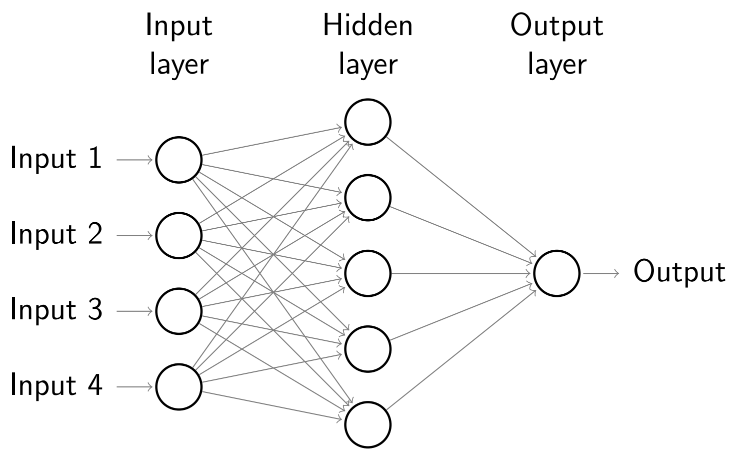

Figure 4: Single Output Feedforward Network

feedforward network is that of Fido. Sensor inputs such as light and sound are sent into the input layer. After the signals have passed through to the output layer, the outputs from the output layer are sent to outputs such as motors, LEDs, and buzzers. The purpose of training a neural network is to make input values correspond to the correct output values, depending on the desired behavior. An example could be making Fido drive when light is applied. The way to give a neural network desired behavior is by controlling the weights of each neuron using a learning algorithm.

3 Learning Implementation

4 Simulation

The robot model chosen for simulation was modeled after easily producible robots on the same scale. The software driving the simulation was intended to be portable enough to work on a hardware implementation, and the model facilitates this goal as well. Additionally, the robot model would have to be easily trainable and debuggable when implemented in hardware; use of a Geiger counter as an input would be unfavorable. Lastly the sensors and design chosen had to facilitate the concept of natural learning, modeling after nature to some degree.

4.1 Robot Inputs and Outputs

Multiple inputs were modeled for simulation with outlets for control both by a human operator using sliders and by programmed handlers using a bridge class. A microphone and light sensor were chosen as clear, human modifiable inputs that model after nature and could easily be used for reinforcement training. An infrared light sensor was added as another easily controller variable in a testing setup: a human operator could easily bring closer and farther an IR LED for purposes of training. Additionally sensors for battery level, three axes of accelerometers, and three axes of gyroscopes were added as more complex inputs for Fido to master.

4.2 Implementation and Kinematics

5 Results

Results, testing, and applications go here.

5.1 Training Methods

5.2 Findings

5.3 Further Applications

6 Discussion

7 Conclusion

Restate, discuss further study, improving experimentation, etc.

References

Dini, S., & Serrano, M. (2012). Combining q-learning with artificial neural networks in an adaptive light seeking robot.

Dudek, G., & Jenkin, M. (2000). *Computational principles of mobile robotics.* New York, NY, USA: Cambridge University Press.

Gaskett, C., Wettergreen, D., & Zelinsky, A. (1999). Q-learning in continuous state and action spaces. In *Australian joint conference on artificial intelligence* (pp. 417–428).

Kim, D. S., & Papagelis, A. J. (n.d.). *Multi-layer perceptron: Artificial neural networks.* `http://www.cse.unsw.edu.au/ cs9417ml/MLP2/`.

MacLeod, C. (2010). *An introduction to practical neural networks and genetic algorithms for engineers and scientists* (Tech. Rep.). Robert Gordon University.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences.