

**“Exploring Artificial Neural Networks with CartPole-v0 Solution”**

An Honors Thesis

by

**Seth A. Law**

California, Pennsylvania

2018

California University of Pennsylvania

California, Pennsylvania

We hereby approve the Honors Thesis of

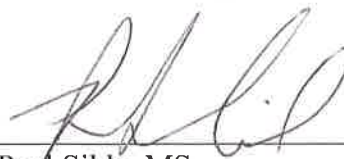
**Seth A. Law**

Candidate for the degree of Bachelor of Science

Date


Faculty

4-18-2018



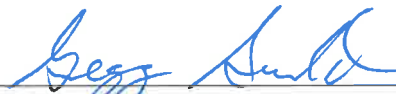
Paul Sible, MS  
Honors Thesis Advisor

4/18/18



Weifeng Chen, PhD  
Second Reader

4-18-18



Gregg Gould, PhD  
Honors Advisory Board

4/18/18



Craig Fox, PhD  
Associate Director, Honors Program

18 April 2018



M. G. Aune, PhD  
Director, Honors Program

## Abstract

Artificial neural networks are a trending topic in computer science and machine learning. Rooted in today's current understanding of how the brain works, artificial neural networks allow machines the capability to essentially learn and decide for themselves. Improvements in technology have provided the means for a proposition between finite automata and neural activity to be realized. Extensive research has been conducted to understand the true potential of this realization, and the results of such research show great promise. To better understand artificial neural networks, their fundamental properties were explored and applied to an existing solution to a problem on OpenAI's website called CartPole-v0.

## Table of Contents

Understanding the Nervous System.....	1
Foundation of Artificial Neural Networks.....	4
Teaching with Mathematics.....	8
OpenAI and “Cartpole-v0”.....	9
“Cartpole-v0” Solution in Python.....	11
Solution Code and Conclusion.....	13
Code Appendix.....	14
References.....	19

## Understanding the Nervous System

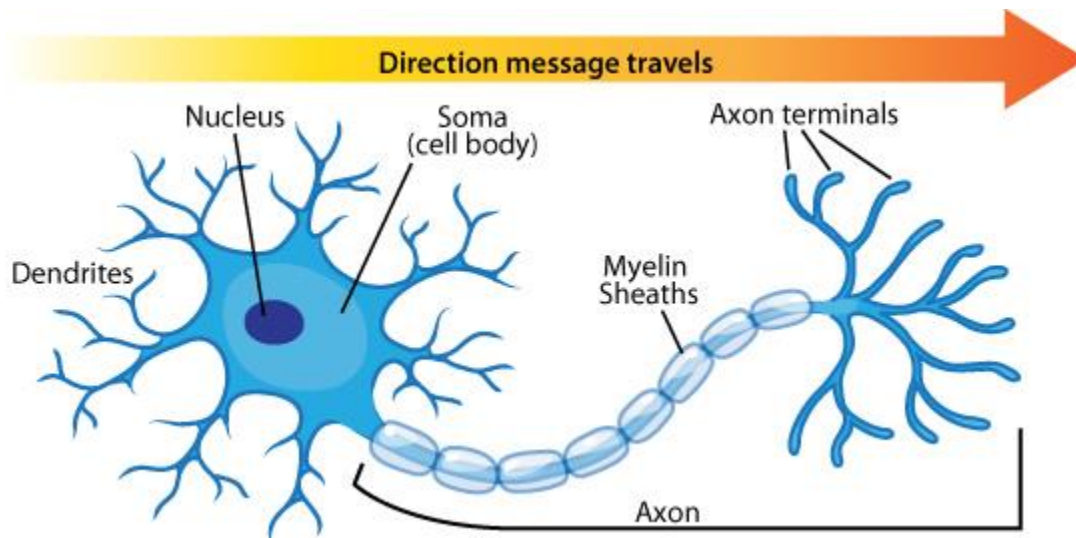
Inside our head is the most complex system known to mankind. Suspended comfortably inside our skull, our brain is currently processing vast amounts of information from various internal and external sources to keep us conscious. It defines who each of us is as a person and what we are as a species. It controls everything about us and is responsible for what has been done, what will be done, and what will continue to be done as long as humans are around. A mass of tissue the size of approximately two fists, weighing anywhere between two and a half to three pounds is *everything*.

The brain is an instrumental component of the nervous system. The human nervous system is an incredibly elaborate collection of nerves stemming from our brain that expands to every part of the body [1]. The central nervous systems primary function is to send signals from one part of the body to another and receive some sort of feedback. The way the nervous system does this is through special cells known as neurons. Neurons are unique from other cells in a variety of ways, but the most noteworthy difference is the ability to communicate through a space called a synapse. The synapse is a structure that allows neurons to pass signals amongst each other via electrical currents or with chemicals called neurotransmitters [1]. The ability of these cells to communicate using electrical signals was a momentous discovery made by observing the structure of nerves underneath a microscope.

Figure 1 is a diagram illustrating the key features of a standard neuron. Emanating from the cell body the dendrites are seen; long finger-like structures that branch out creating dendritic trees. This dendritic tree is responsible for receiving signals sent from

the axons of other neurons and determining whether the neuron should send a signal down its own axon.

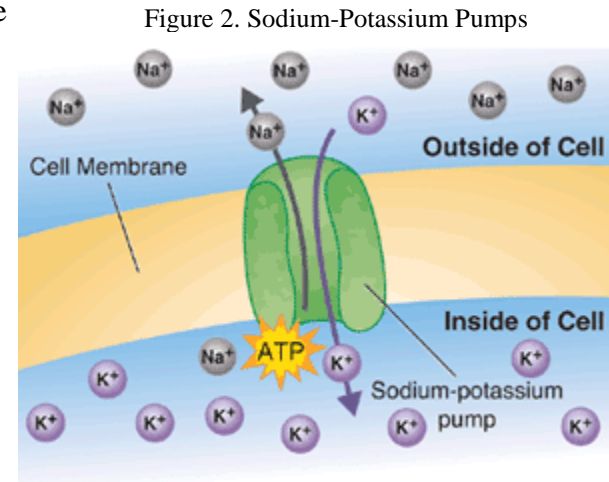
Figure 1. Standard Neuron



Neurons are very efficient at transferring signals and processing feedback. Every neuron comes equipped with a membrane that maintains a voltage gradient. This charged membrane possessed by the neuron is electrically excitable and thus is capable of being influenced by action potential.

A neuron is analogous to a battery. By itself, a battery is nothing more than a collection of chemicals holding stored energy just waiting to be used. In the region surrounding the neuron, positive sodium ions are present while positive potassium ions are enclosed inside the neuron. The presence of more sodium ions outside the neuron than potassium ions inside the neuron means the net charge of the neuron is negative. A neuron in this state is known as polarized [2]. Figure 2 shows a polarized neuron that is electrically charged by proteins called the sodium-potassium pumps. They ride the membrane of the neuron pumping an unequal amount of sodium and potassium ions to and from the neuron. This uneven distribution of ions creates the electrical gradient. To

even out this gradient, ions must be able to pass from the environment into the neuron through channels in the membrane. These channels open from various stimuli depending on the primary function of the neuron. To send a significant signal down an axon, there must be a powerful trigger that opens the numerous voltage-gated channels. If the trigger crosses a certain threshold, then the action potential is realized, and ions flood the neuron rapidly depolarizing it. That local change in current travels down the axon to some designated neuron that will respond accordingly [2]. The neuron will eventually return to a polarized state via the sodium-potassium pumps and the cycle can be repeated.



Studies of the human brain have estimated that there are over 86 billion neurons present [3]. The amount of interconnectivity amongst neurons in the typical human creates a network of unimaginable complexity. This network of communication is where the true power of the neuron lies. Between each junction of axon and dendrite among connected neurons exists an ever so minute gap known as a synapse. Anywhere between 20 to 40 nanometers across, the synapse is a fundamental element of neurotransmission [3]. Communication is achieved when a presynaptic neuron passes a signal across the synapse to a postsynaptic neuron. Chemical synapses use neurotransmitters located in the presynaptic membrane of the neuron to bind to receptors located on the membrane of the postsynaptic neuron. Chemical synapses have a variety of classifications depending on the type of neurons at play. Neurotransmitters themselves are vastly complex and their

effects on the post synaptic neuron are intricate. Chemical synapses are responsible for a major portion of activity in a typical biological neural network.

Electrical synapses behave in a more straightforward manner. Thus, these electrical synapses are inherently simpler than chemical synapses. The exclusion of neurotransmitters from the communication process means that electrical synapses are less varied and more resistant to external influences. Electrical synapses transmit signals almost instantly, making them perfect for scenarios in which a rapid response is required. Transmission speeds are so fast that neurons can even fire synchronously. Electrical synapse responses occur quickly, are bidirectional if need be, and produce simple behavior [4].

The rapid development of technology has unearthed a striking link between neural behavior and the computational theory of finite automata. Could a machine be able to model neurological activity? To answer that question, artificial neural networks were conceptualized by Walter Pitts and Warren McCulloch in 1943.

### **Foundations of Artificial Neural Networks**

The evolution of artificial neural networks traces its roots to a paper written by Warren McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity” laid the foundations from which neural networks were built. McCulloch and Pitts demonstrated that “neural events and the relations among them can be treated by means of propositional logic [5].” They found that the behavior of every net without circles can be defined in their logical calculus, but certain key assumptions



needed to be made. These are the following assumptions made on neural nets that do not cycle (without circles):

1. The activity of the neuron is an “all-or-none” process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
3. The only significant delay within the nervous system is synaptic delay.
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
5. The structure of the net does not change with time.

To summarize the following assumptions, artificial neural nets are made up of an arbitrary amount of nodes (neurons) whose network structure is immutable. The only transmission delay between nodes is synaptic, which is the relative distance between nodes. The activity of an individual node is all or nothing. A node that has been triggered by the network either transmits or prohibits subsequent signals. There are no partial transmissions or partial blockages; all activity after excitation is absolute. From these cardinal assumptions, Pitts and McCulloch used extensive calculus to translate neural activity into some relatively straightforward mathematics. Many papers have been

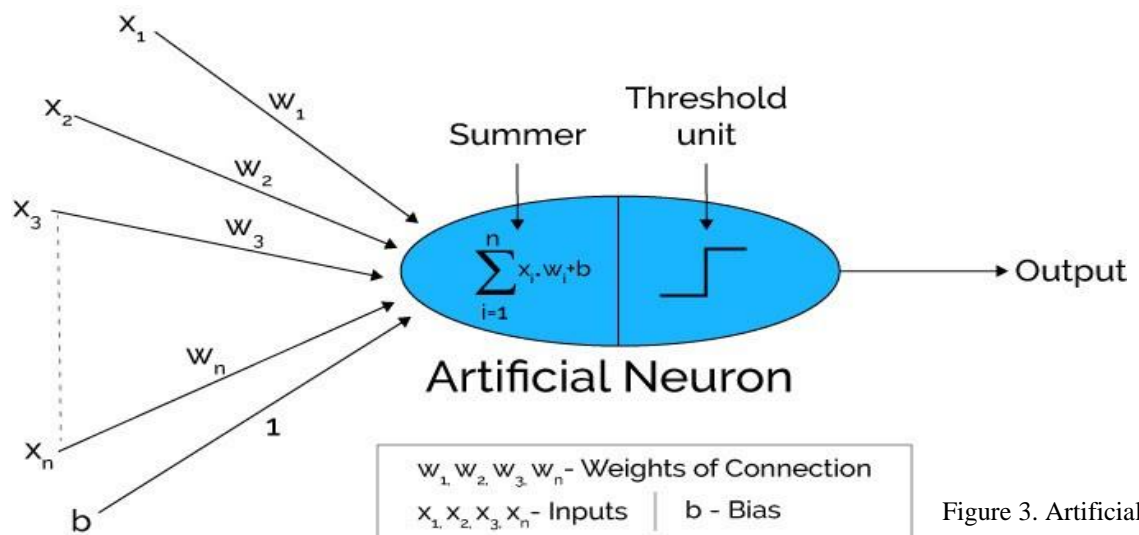


Figure 3. Artificial Neuron

published and much research has been done regarding different network models, mathematical properties, and neural anomalies since the Pitts-McCulloch paper. However, the interest of this paper lies solely in understanding and applying a basic artificial neural network with only the essential mathematics needed to solve a problem. Figure 3 illustrates the best way to understand the model and the mathematics used in an artificial neural network.

Figure 3 is the depiction of a standard node/neuron in an artificial neural network. Neurons begin with an arbitrary number of inputs represented by  $x_1$  through  $x_n$ . The origin of the inputs varies depending on the location of the neuron within the network. If the neuron is part of the input layer, or the beginning layer of the network, the input is typically provided by a user of the network. If the neuron is not part of the input layer, the inputs are provided or inhibited by neurons in a prior layer. The values of these inputs are then multiplied by a weight represented by  $w_1$  through  $w_n$  in Figure 3. Weights will be discussed in further detail later. After all the inputs have been multiplied by their respective weights, the determined values of all inputs will then be summed (Summer in Figure 3). To this sum, a bias will also be added. Like weights, a bias will be discussed later. The first assumption established earlier was that the activity of a neuron is all or nothing. When all the input values are determined and summed along with the bias, that subsequent value could be any possible number. To realize this assumption, the summed value will be passed through an activation function labeled Threshold unit in Figure 3. This activation function will determine if the neuron fires or not.

A typical activation function is a sigmoid function. The sigmoid function is bounded, differentiable, and is defined for all real input values. Expressed as  $\frac{1}{1+e^{-x}}$  the sigmoid function will take the value of the neuron's sum,  $x$ , and produce a number between 0 and 1. This function works extremely well as an activation function since no matter what real value the summation produces, once it's passed through the sigmoid function, only a value between 0 and 1 will be produced. This translates accordingly to the neuron firing or not.

Nodes of an artificial neural network are then grouped into layers. These layers are known as the input layer, one or more hidden layers, and an output layer. The nodes in each layer are connected to each node in a subsequent layer creating a network such as Figure 4.

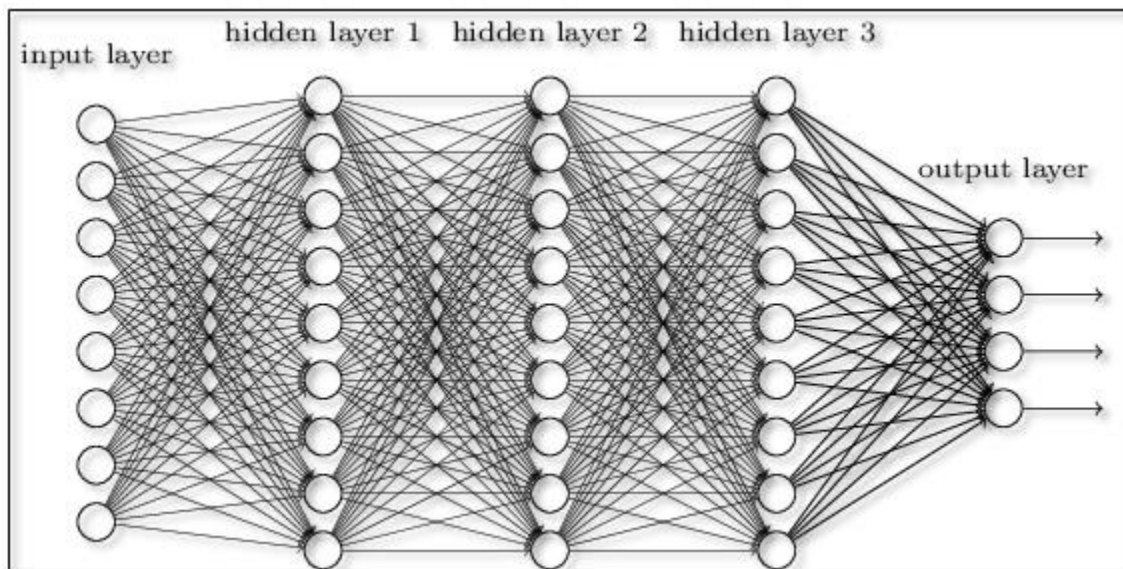


Figure 4. Artificial Neural Network

## Teaching with Mathematics

Perhaps the most charming aspect of artificial neural networks is their capability to “learn”. However, the term “learn” is rather misleading. With the increased amount of automation in today’s workforce, giving machines the ability to learn can be a concerning thought. Fortunately for us, training an artificial neural network is more akin to teaching a machine to produce correct results rather than it consciously learning on its own. This is accomplished by some extremely clever calculus and linear algebra.

Teaching an artificial neural network is fundamentally a challenge in optimization. The network starts with input data and some randomly generated weights and biases. These random weights and biases will at first produce results that are unpredictable and largely incorrect. To tweak these weights and biases, a cost function is defined. The purpose of the cost function is to measure how well the network is doing with the current weights and biases. The cost function will take the current weights and biases as its input and produce a single output value. This single output value is an indication of how well the network is performing. When this output is large, the current values of the weights and biases are causing the network to perform poorly. Lowering the output value of the cost function is done by minimizing the cost function. By minimizing the cost function, an artificial neural network learns.

Minimizing the cost function is accomplished by gradient descent. If a multi-variable function  $F(x)$ , in our case the cost function, is differentiable at some point i.e. the random weights and biases defined, then  $F(x)$  will decrease fastest in the direction of the negative gradient of  $F$  [6]. What is essentially transpiring is the slope of the cost function is determined at some instance and the goal is to find out what direction to travel to get

that slope as flat as possible. Given the current output of the cost function, adjustments are made to the weights and biases and another output is produced. That output is then analyzed, the weights and biases are adjusted further, and the direction to travel is determined again. Repeating this process hundreds, thousands, even millions of times refines the weights and biases to optimal values; thus, an artificial neural network is trained.

Another important topic is how the gradient of the cost function is computed. This is known as backpropagation. In 1986, “Learning representations by back-propagating errors” by David Rumelhart, Geoffrey Hinton, and Ronald Williams produced a learning procedure coined backpropagation that illustrated how quickly the output of the cost function changes with respect to altering the weights and biases of a network [7].

Alterations to weights and biases have profound effects on the network that ultimately ripple out and change aspects of the network as a whole. A minor adjustment in one area will indefinitely alter other areas. The underlying calculus behind it is well beyond the scope of this paper, but it deals with the partial derivative of the cost function with respect to the weights and biases. The important aspect to take away from backpropagation is that certain weights and biases are more influential on the network than others.

### **OpenAI and “CartPole-v0”**

In 2015, Elon Musk, the CEO of SpaceX, along with a handful of other investors pledged over \$1 billion in funds towards the development of artificial intelligence (AI). This manifested itself into a non-profit organization known as OpenAI. Musk believed at the time that AI was humanity’s greatest threat, so the goal of OpenAI was simple;

provide a free, community-based sanctuary where AI could be developed and researched with a focus on “positive human impact” [8]. He has since left the organization due to possible conflicts of interest, but OpenAI is still thriving. In 2016, OpenAI released its first platform for machine learning called “OpenAI Gym”.

Gym is essentially a toolkit that provides users with problems called environments. Users can then freely test and develop learning algorithms on a plethora of environments. These environments make no assumptions about algorithms or model structures, share a common interface allowing for general algorithms to be ported between environments, and are compatible with any numerical computation library [9]. The environments vary in complexity, spanning from simple locomotion simulations to full on Atari emulations like *Ms. Pacman* and *Pitfall*. All the environments have a goal which serves as a purpose for training models. Users are then able to freely post their solution to the OpenAI website to show off how effective their training methods were.

The environment chosen for this paper is known as “CartPole-v0”. The problem was derived from a paper titled “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems” written by researchers of the Institute of Electrical and Electronics Engineers (IEEE) [10]. It begins with a pole attached to a cart which moves along a frictionless path. The pole begins in the upright position and is bound to the cart by an un-actuated joint i.e. it won’t move unless a force is acted upon it. The goal is simple; prevent the pole from falling over. If the pole is more than  $15^\circ$  from vertical or the cart has moved more than 2.4 units from the center, then the game is over.

## **“Cartpole-v0” Solution in Python**

Harrison Kinsley, who will be referred to from now on by his online pseudonym Sentdex, is a self-taught programmer and entrepreneur that has created numerous websites and YouTube videos dedicated to educating others on a variety of topics regarding the popular programming language Python. Sentdex’s resources on the topic of artificial neural networks have been invaluable throughout this research process and combing through his code was a delight. His code served as a foundation from which this solution was built, and the many hours spent watching his videos and reading his website has been a prime inspiration for this solution. The code is separated into three distinct methods; one for collecting training data, one for defining the neural network model, and one for training the model. All of this is made possible by three Python libraries; `gym`, `numpy`, and `tflearn` [11].

The first method `training_games()` creates a set amount of games, plays the games with random action input, and stores all the necessary data collected from each game session. This method utilizes `gym` to create the “CartPole-v0” environment and manage the data collection. A `score_requirement` is created so that information is only collected from games with optimal scores. This is necessary to ensure the neural network is being trained with the best possible data collected. Another important note about this method is that it converts the collected game data into what is known as a one-hot format. The cart in this environment can only move left or right. If it chooses to move right, then that choice is ‘hot’. It’s very similar to binary in which a one is ‘hot’ and a zero is ‘cold’. The data is encoded in this manner so that it can be easily translated by the neural network later.

The next method, `neural_network_model`, takes one parameter, `input_size`, which is used to create the input layer for the neural network. `tflearn` utilizes two methods, `fully_connected` and `dropout` to define and create the entire network instead of manually creating this data structure from scratch. Each line of `fully_connected` is passed the size (number of nodes in the layer) and the type of activation function used. `dropout` is passed the object which invokes `fully_connected` and then the keep probability. The keep probability is a threshold which each single node must cross to activate. The final `fully_connected` establishes the output layer and a part of the cost function. All of this is then passed to a `DNN` method which wraps up creation of the network and stores it in an object called `model`.

The final method `train_model` is admittedly convoluted, but it uses the `numpy` library to perform matrix manipulation on the training data and essentially performs reinforcement training. With two lists, one X, one Y, `train_model` acts like a cost function by comparing the two lists, one being what was observed, the other being what should have been observed, and performing alterations to the weights and biases based on the cost of the differences.

By calling all three of these methods, the network is built, provided data, and subsequently trained. All that is left is to see how well it performs by invoking the environment again and allowing the network to predict what actions to take. “CartPole-v0” is considered solved if the average score is over 195 across 100 trials. With samples



from 100,000 training games and a score requirement of 100, this program averages a score of 200 and thus is considered solved.

### **Solution Code and Conclusion**

Now that technology has finally caught up to theory, artificial neural networks are becoming a fascinating paradigm to follow. A team at OpenAI created a bot for the popular game *Dota 2* that trained entirely against itself for roughly three months. It was then released upon professional players in a one-on-one scenario and crushed each and every one of them [12]. A bot had developed greater skill in three months than these players had developed over the course of years. MNIST (Modified National Institute of Standards and Technology) created a substantial database of pixel data from which networks have trained from in order to recognize handwritten digits. There are even reports of networks being able to steer vehicles, recognize faces, and even diagnose certain cancers based on cell shape information.

In a world where information and data are everything, artificial neural networks will indefinitely find a niche in future computing. Although artificial neural networks have been criticized for the inability to solve computationally difficult problems and the need for massive amounts of data/computing power to train, they should not be underestimated or discarded. It appears artificial neural networks are simply a piece of the much larger puzzle of technological enlightenment. OpenAI's *Dota 2* bot has shown that artificial neural networks can easily surpass the capabilities of a human. The future of artificial neural networks is as mystifying as it is potentially horrifying, but it certainly is incredible what technology is capable of.

Code Appendix

```

import gym #Toolkit for learning algorithms
https://github.com/openai/gym
import random #Pseudo-random number module
https://docs.python.org/3.4/library/random.html
import numpy as np #Package for scientific computing
http://www.numpy.org/
import tflearn #Deep learning library from Tensorflow
https://github.com/tflearn/tflearn

from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
from statistics import mean, median
from collections import Counter

#Learning rate of neural network
LR = 1e-3

#Define our environment for the network, initialize it
environment = gym.make('CartPole-v0')
environment.reset()

#Timesteps for environment
time_steps = 500

#Keep the data from games with this score or higher
score_requirement = 70

#Number of initial games played for training data
initial_games = 1000

#Target number of timesteps
goal_steps = 500

#Play some games with random behavior for training data
def training_games():

    #[Observations, Moves]
    training_data = []

    #Every score achieved in training games
    scores = []

    #Scores that met our requirements
    accepted_scores = []

    #Begin simulating training games
    for _ in range(initial_games):

        score = 0

        #[Cart Position, Cart Velocity, Pole Angle, Pole Velocity at
Tip]
        game_memory = []

```

```

#List containing each value of the previous observation
prev_observation = []

#environment.render()

#For each time step, do a random action
for _ in range(time_steps):

    #Actions are 0 (push left) or 1 (push right)
    action = environment.action_space.sample()

    '''
    Step returns 4 values: observation(object), reward(float)
    done(boolean), and info(dictionary). This is the agent-
environment
    loop. Each time step, the agent chooses and action and the
    environment returns an observation and a reward
    '''
    observation, reward, done, info = environment.step(action)

    #If our previous observation was successful
    if len(prev_observation) > 0:

        #Append to game_memory the previous observation
        #and the action taken by the agent
        game_memory.append([prev_observation, action])

    prev_observation = observation
    score+=reward

    #If done returns true, the game has finished
    if done:
        break

#If a score reached our score requirement
if score >= score_requirement:

    #Add it to the list of accepted scores
    accepted_scores.append(score)

    #Convert game_memory to one-hot format for output layer
    for data in game_memory:

        if data[1] == 1:

            #Action agent took (right)
            output = [0,1]

        elif data[1] == 0:

            #Action agent took (left)
            output = [1,0]

    #Store data from training games into training_data list
    training_data.append([data[0], output])

```

```

        environment.reset()
        scores.append(score)

    """
    Use to view stats about training data
    print('Highest score:', max(scores))
    print('Average accepted score:', mean(accepted_scores))
    print('Median score for accepted scores:', median(accepted_scores))
    print(Counter(accepted_scores))
    """

    return training_data

training_games()

#Create model for neural network
def neural_network_model(input_size):

    network = input_data(shape=[None, input_size, 1], name='input')

    network = fully_connected(network, 64, activation='relu')
    network = dropout(network, 0.8)

    network = fully_connected(network, 128, activation='relu')
    network = dropout(network, 0.8)

    network = fully_connected(network, 256, activation='relu')
    network = dropout(network, 0.8)

    network = fully_connected(network, 128, activation='relu')
    network = dropout(network, 0.8)

    network = fully_connected(network, 64, activation='relu')
    network = dropout(network, 0.8)

    network = fully_connected(network, 2, activation='softmax')
    network = regression(network, optimizer='adam', learning_rate=LR,
loss='categorical_crossentropy', name='targets')
    model = tflearn.DNN(network, tensorboard_dir='log')

    return model

def train_model(training_data, model=False):

    #Reshaping training data with numpy (not really sure)
    X = np.array([i[0] for i in training_data]).reshape(-
1, len(training_data[0][0]), 1)

    #Filling Y with target data
    Y = [i[1] for i in training_data]

    if not model:
        model = neural_network_model(input_size = len(X[0]))

```

```

'''
This model.fit passes 4 parameters:
1: X input data as a dictionary
2: Y target data to train model
3: Number of epochs (backpropagation after each epoch)
4: Displays accuracy at every step
'''

model.fit({'input': X}, {'targets': Y}, n_epoch=7,
show_metric=True)
return model

#Perform training_games and subsequent training
training_data = training_games()
model = train_model(training_data)

#Lists for new scores and choices
#made by the model
scores = []
choices = []

#Functions similarly to training_games
for each_game in range(100):

    score = 0
    game_memory = []
    prev_obs = []
    environment.reset()

    for _ in range(goal_steps):

        #environment.render()

        #Start game off with random action
        if len(prev_obs) == 0:

            action = random.randrange(0,2)

        #Otherwise use model's prediction
        else:

            action = np.argmax(model.predict(prev_obs.reshape(-
1,len(prev_obs),1))[0])

        #Save action taken by model
        choices.append(action)

        new_observation, reward, done, info = environment.step(action)
        prev_obs = new_observation
        game_memory.append([new_observation, action])
        score+=reward

        if done: break

    scores.append(score)

```

```
#Useful stats for post training
print('Average Score:',sum(scores)/len(scores))
print('choice 1:{}'.format(choices.count(1)/len(choices)),choice
0:{}'.format(choices.count(0)/len(choices)))
```

## References

- [1] Kandel ER, Schwartz JH, Jessel TM, eds. (2000). "Ch. 2: Nerve cells and behavior". *Principles of Neural Science*. McGraw-Hill Professional. [ISBN 978-0-8385-7701-1](#).
- [2] Green, Hank. (2015, March 2). Crash Course the Nervous System. *The Nervous System, Part 2 – Action! Potential!: Crash Course A&P #9*. Retrieved from [https://www.youtube.com/watch?v=OZG8M\\_ldA1M](https://www.youtube.com/watch?v=OZG8M_ldA1M)
- [3] Sukel, Kayt. (2011, March 15). The Dana Foundation. *The Synapse – A Primer*. Retrieved from <http://www.dana.org/News/Details.aspx?id=43512>
- [4] Hormuzdi, Sheriar G. et al. (2004, March). *BioChimica et Biophysica Acta – (BBA) Biomembranes. Electrical synapses: a dynamic signaling system that shapes the activity of neuronal networks*. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0005273604000410>
- [5] Pitts, Walter. McCulloch, Warren S. (1943). Society for Mathematical Biology. *A Logical Calculus Of The Ideas Immanent In Nervous Activity*. Retrieved from <https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf>
- [6] 3Blue1Brown. (2017, October 16). Neural Networks. *Gradient descent, how neural networks learn | Chapter 2, deep learning*. Retrieved from <https://www.youtube.com/watch?v=IHZwWFHWa-w&t=336s>
- [7] Nielson, Michael. (2017, December). *Neural Networks and Deep Learning*. Retrieved from <http://neuralnetworksanddeeplearning.com/index.html>

- [8] British Broadcasting Company. (2015, December 12). British Broadcasting Company. *Tech giants pledge \$1bn for 'altruistic AI' venture, OpenAI*. Retrieved from <http://www.bbc.com/news/technology-35082344>
- [9] OpenAI. (2016). OpenAI. *Getting Started with Gym*. Retrieved from <https://gym.openai.com/docs/>
- [10] Barto, AG. Sutton, RS. Anderson, CW. (1983). IEEE Transactions on Systems, Man, and Cybernetics. *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem*. Retrieved from <https://gym.openai.com/envs/CartPole-v0/#barto83>
- [11] Kinsely, Harrison. (2017, March 13). Sentdex. *Using a neural network to solve OpenAI's CartPole balancing environment*. Retrieved from <https://pythonprogramming.net/openai-cartpole-neural-network-example-machine-learning-tutorial/>
- [12] OpenAI. (2017, August 6). OpenAI. *More on Dota 2*. Retrieved from <https://blog.openai.com/more-on-dota-2/>

FIGURE 1 <https://askabiologist.asu.edu/neuron-anatomy>

FIGURE 2 <https://socratic.org/questions/what-is-the-role-of-potassium-in-muscle-contraction>

FIGURE 3 <https://medium.com/@xenonstack/overview-of-artificial-neural-networks-and-its-applications-2525c1adff7>

FIGURE 4 <http://www.thewindowsclub.com/deep-learning-and-neural-network>



Seth Law

Major in Computer Science, Minor in Mathematics

Committee Members: Paul Sible, Weifeng Chen, Gregg Gould

Artificial Neural Networks, Machine Learning, Python, Neurons

Title: Exploring Artificial Neural Networks with “CartPole-v0”

Solution