Scary Sonata

Using the Hebbian Learning Rule

Austin Hufstetler | CS4820 | October 14, 2017

Contents

Description and Objectives

The premise of the program is to train the girl to fear a song (Beethoven's 5th symphony). She is naturally scared of the red ogre that is flying above her, however she is not naturally scared of the song. To train her to fear the song, we must use the classical conditioning (in the form of the Hebbian Learning Rule). When the ogre appears, we play the song. Eventually the girl becomes scared when the song is playing, even if she cannot see the ogre.

Motivations

I thought fear was one of the most logical and easy to understand forms of classical conditioning. Many horror films use signature music to incite fear into their audience; whenever a certain song is played, the audience knows the villain is close. This common practice is where I got the idea. Granted, the monster could have been a bit scarier, but I wanted to keep the "cartoonish" feel of the entire scenario.

How to Run the Program

You won't need to do anything with the girl and apple objects. You only need to change settings on the music player and ogre.

To make ogre appear: Right click *red balloon* and click *makeVisible()*

To make ogre disappear: Right click *ogre* and click *makeInvisible()*

To turn music player on: Right click *music player* and click *turnOn()*

To turn music player off: Right click *music player* and click *turnOff()*

To pause music when program pauses: Right click *music player* and click *pauseSong()* (the music will resume when program starts)

To test the training, you will probably want to test the ogre and music player separately before training. You will see the girl runs when the ogre appears, but she just smiles when the music is playing and ogre is invisible.

To train, simply turn the music on and make the ogre appear at the same time. After a few seconds make the ogre invisible and turn the music off. Now, turn only the music on and the girl should be scared.

It is recommended to keep the speed in the center.

Architecture and Design

Overview

There are two input neurons (music player and ogre) and one output neuron (girl). There are two weights, one from the music player to the girl and one from the ogre to the girl.

Classes

MyWorld:

Description:

The world where all other objects are placed.

Methods:

MyWorld()

Constructor.

Creates the park and places the ogre, girl, music box, and apple in it. Also calls the initialize method from the Controller class.

RedOgre:

Description:

Serves the purpose to scare the little girl. Spins around in a circle when visible. Whenever invisible, it becomes a balloon. The monster is from Japanese folklore and is called an Oni.

Methods:

act()

makeVisible()

Changes the balloon to the ogre

isVisible()

Returns zero if the ogre is invisible and one is visible

makeInvisible()

Makes the ogre invisible

changeVisibility(int x)

Changes the visibility of the ogre

MusicPlayer:

Description: Plays Beethoven's 5th symphony when on.

Methods:

act()

turnOn()

Sets "on" to 1 and takes effect when scenario is unpaused

turnOff()

Sets "on" to 0 and takes effect when scenario is unpaused

isOn()

Returns if music player is on or off (0 for off and 1 for on)

changeOn(int x)

Changes state of on

playSong()

Plays the song

stopSong()

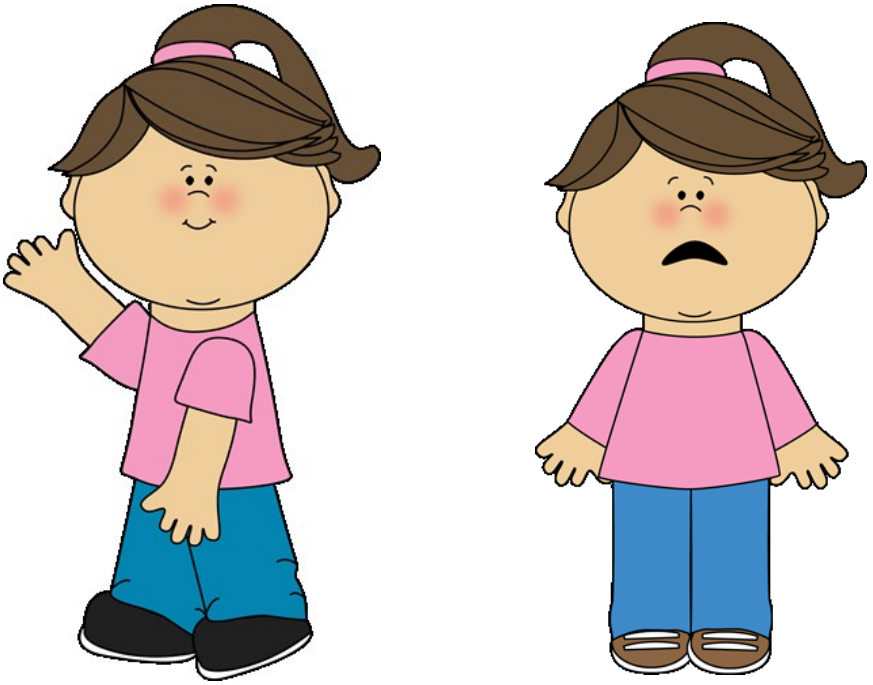
Directly stops song.

pauseSong()

Pauses music while scenario is not running. Music is resumed after program turned on.

Girl:

Description: The girl who will be trained.



Methods:

act()

makeRun ()

Sets “on” to 1 and takes effect when scenario is unpaused

stopRun()

Sets “on” to 0 and takes effect when scenario is unpaused

getRun()

Returns if music player is on or off (0 for off and 1 for on)

changeStatusRun(int x)

Used in the Hebbian algorithm to change status of running

Controller:

Description: Serves as the connection between the Hebbian class and the objects in the world. Represented as an apple on the tree.

Methods:

act()

initialize(Girl g, RedOgre r, MusicPlayer m)

Acts as a connection between the Hebbian algorithm and world. Initializes the Girl, Red Ogre, and Music Player.

Hebbian:

Description: Contains the code that uses the Hebbian Learning Rule to each the little girl to fear the music.

Methods:

Hebbian()

Constructor

activate(Girl g, RedOgre r, MusicPlayer m)

This is step 1 in the Hebbian algorithm. Calculates the activated value and sends it to the step function. That value is then sent to the method that either makes the girl run or stops the girl from running.

train()

This is step 2 in the Hebbian algorithm. Calculates the new weights.

stepFunction(double x)

Returns 0 if value is less than 0 and 1 if greater than 0

Results

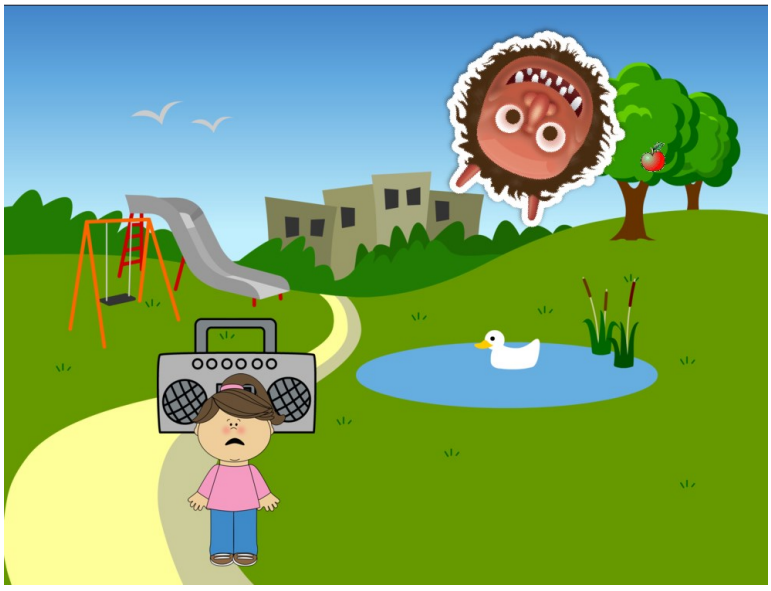
The training was successful. After several seconds with both the music playing and the ogre visible, the girl becomes frightened of the music.

State 1:

Ogre is invisible and music is not on.

State 2:

Ogre is visible and music is not on.

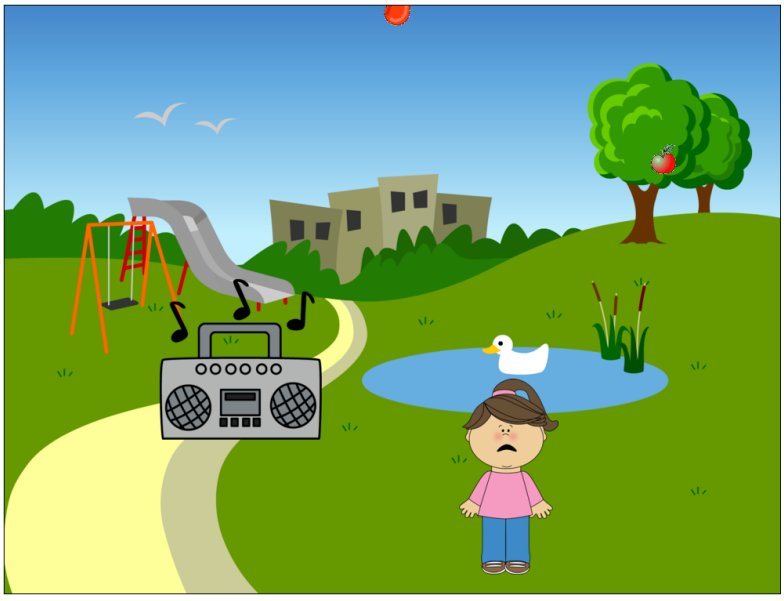


State 3 (Pre-training):

Ogre is invisible and music is on.

State 3 (Post-training):

Ogre is invisible and music is on.



Discussion

Strengths:

- Effectively trains the girl to fear the music
- Trains in a timely manner
- The Hebbian algorithm is controlled by a single class (Controller class shown as an apple). This object deals with all calls to the Hebbian algorithm

Weaknesses:

- The Hebbian algorithm is intertwined with the world itself; it only works with Girl, RedOgre, and MusicPlayer objects. The code would need to be modified to allow different actors.

Desired Features:

- Would like to have added more songs, possibly making the girl scared of certain songs and not others.
- Allow the user to change the characters (i.e change the monster and change the girl)

Mechanisms of ANNs

Supervised with feed-forward/back-propagation

Supervised learning provides some training set of known values. We “teach” the program until a certain level of accuracy is satisfied. It is called “supervised” because we provide the known values, as well as telling it when it should stop learning.

Feed-forward refers to the act of sending signals in from the input to the output, usually through a set of hidden neurons.

Backpropagation refers to the use of traversing an ANN backwards to calculate how the weights should be changed due to some error.

These two were combined in our first project. You send the values through the ANN and find the error when comparing to the expected output. You then back propagate and change the weights according to how much it contributed to the overall error.

Unsupervised with Hebbian learning

Unsupervised learning doesn't need a teacher. The algorithm learns just how a brain would learn.

Hebbian updates the weights of connections based on nearby activated connections. If one connection is activated, its neighbors can also become activated with enough training.

Classical conditioning is a good example of this. An activated stimulus causes a previously inactive stimulus to be active. This is similar to how humans and other animals learn.

Unsupervised with competitive learning

As mentioned above, unsupervised learning doesn't need a teacher, and it mimics how brains learn.

Competitive learning is a winner take all learning technique. The neurons compete to be active and a single neuron is chosen to be active based on some criteria (highest value, lowest value, etc).

The Kohonen model is an example of competitive learning. This model causes neurons in the winner's neighborhood to become "excited" but has the opposite effect for neurons far away from the winner. Eventually these groups form clusters which makes this model self-organizing.