Thomas Kelly
15 April 2015
IN3016 Coursework

A Software Agent Model of Operant Conditioning

Introduction

Operant conditioning is a concept that was originally developed by famed behavioral psychologist B. F. Skinner in his 1938 book, *The Behavior of Organisms*. It essentially describes a learning method in which positive behaviors are increased, and negative behaviors reduced, via reinforcement and punishment. In this paper, I will describe a software agent model that mimics the effects of an operant conditioning experiment. I will also compare the results of different testing parameters, to determine if they have any significant effect on learning.

Background

A traditional operant conditioning task consists of four parts: positive reinforcement, negative reinforcement, positive punishment, and negative punishment. For the purposes of machine learning, each of these can be considered a type of reward. Their purpose is to either increase or decrease a specific type of behavior, or action. 'Reinforcement' refers to any reward which increases a correct behavior, while 'punishment' refers to rewards which decrease incorrect behaviors. 'Positive' rewards involve the addition of some stimulus, while 'negative' rewards remove the stimulus. A stimulus itself can come in one of two flavors — appetitive stimuli move the subject closer to its goal state, whatever that may be, while noxious stimuli do the exact opposite.

It should also be noted that negative reinforcement can be achieved in two distinct ways. The first is escape, in which the subject performs a correct behavior that results in the removal of a noxious stimulus. The other is avoidance, in which the subject performs a correct behavior that prevents a noxious stimulus from occurring. It's a subtle difference, but an important one nonetheless.

Domain and Task

This model seeks to demonstrate all five of these reward mechanisms by placing a software agent in a maze of connected rooms. The agent, who shall henceforth be known as Bob, seeks to find a way out of the maze. Bob is very concerned for his own health & comfort, so he also wants to be as well-fed and in as little pain as possible when he reaches the exit. Each room in the maze is initially locked, and contains a task which Bob must complete before the room becomes unlocked and he is allowed to move on to a connected room. These tasks come in two forms, representing the two types of stimuli: appetitive and noxious.

If Bob finds himself in an appetitive task room, he is faced with three buttons and has to learn the correct sequence of three button presses in order to proceed. Whenever a correct button is pressed, Bob received a food pellet, making him less hungry. However, pressing an incorrect button resets the task, forcing Bob to start over.

Upon entering a noxious task room, a barrier surrounds Bob that hinders his movement. The barrier is short enough to jump over, but Bob is a naturally lazy agent and would prefer not to exert himself unless he needs to. However, after a certain amount of time passes, the floor inside the barrier becomes electrified and Bob receives a painful shock. The likelihood of jumping over the barrier drastically increases as Bob

tries to escape the painful stimulus. The next time Bob enters a room similar to this one, he will be a bit more likely to jump right away in order to avoid being shocked.

State Transition Function

A state in this model describes all of the information that Bob can perceive about his environment at a specific point in time. This includes the contextual clues of the current room, essentially letting Bob know whether it is locked or unlocked, and whether its task is appetitive or noxious. Bob can also look around and see how many doorways are on the walls of the room, letting him know how many rooms are connected to this one. He also has a decent memory & sense of direction, so he can tell which connected rooms he has already visited. Finally, Bob is wearing a watch that allows him to tell the current time step, which increments each time Bob makes any task- or movement-related decision.

The actions that Bob is able to take depend first on whether his current room is locked or unlocked. If Bob is in an unlocked room, he must move to one of the connected rooms. If he's in a locked room, he must then determine the type of task in the room and take an action corresponding to this task. All of these actions are probability-based — the probability of taking incorrect actions are decreased, and correct ones increased, by the reward stimulus Bob receives in response to them. Thus the state transition function that decides Bob's next state, given the current state-action pair, is based on probabilities that change dynamically throughout the trial depending on Bob's experiences.

Reward Function

Bob possesses two reward values, hunger and pain, which are affected by his current state-action pair. The hunger value represents Bob's response to appetitive stimuli; it is increased via negative punishment (being hungry) and decreased via positive reinforcement (receiving a food pellet). The pain value is Bob's response to noxious stimuli; it is increased via positive punishment (receiving a shock) and decreased via negative reinforcement (either escape or avoidance of said shock). Lower reward values are therefore considered to be better than higher reward values.

Policy

When Bob first enters the maze, his action-selection policy is based on purely random decisions. Each time he makes a decision, however, he remembers the resulting stimulus, forming associations which allow him to alter the probability that he will make that decision again. In this way, Bob transitions from a random-selection policy to a probabilistic policy as he makes his way through the maze. However, there are also three sub-policies that determine how the probabilities of certain actions are altered.

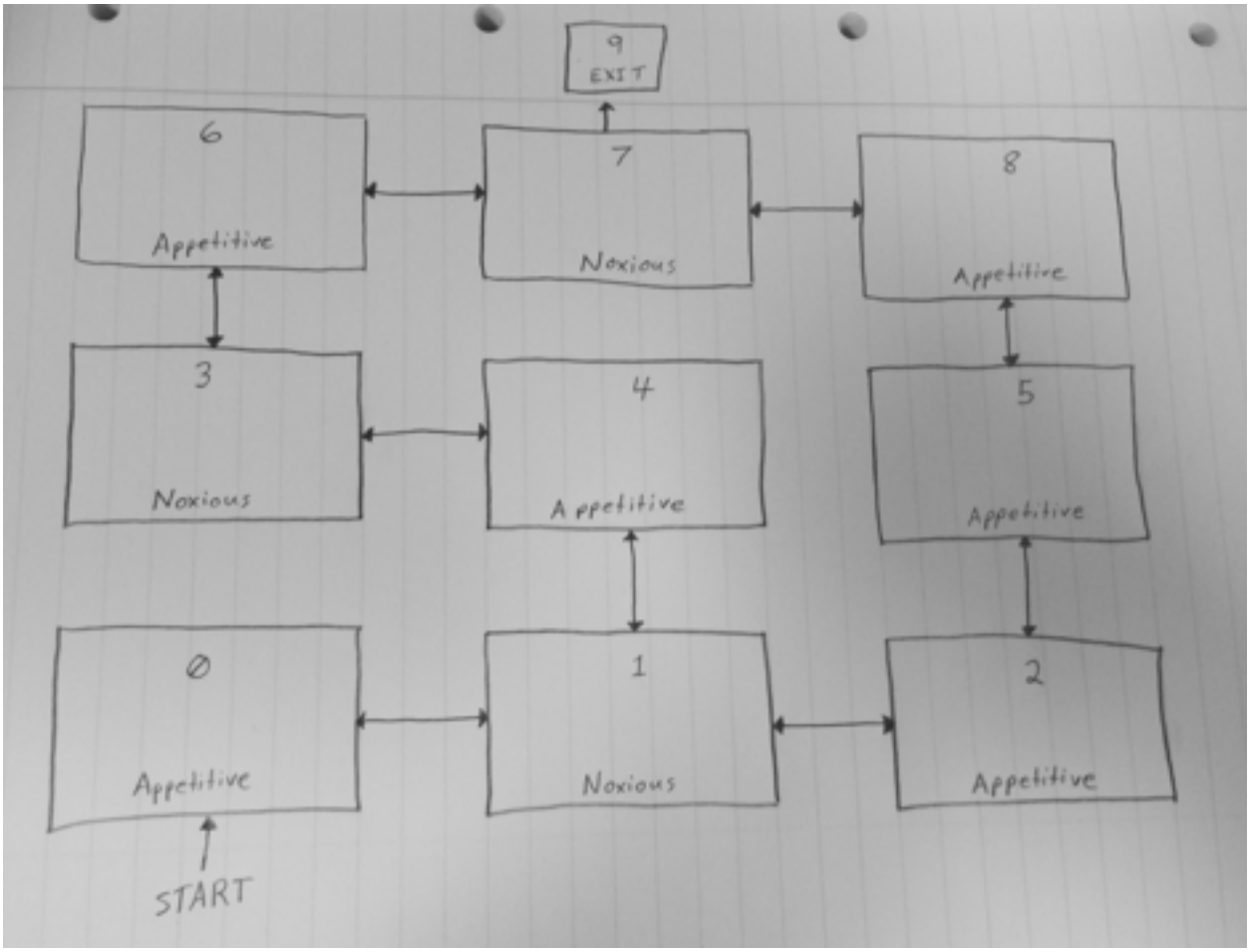
For instance, selecting the correct button in an appetitive room increases the probability of pressing that button at that step in the sequence by 10%, while simultaneously reducing the probability of pressing either of the other two buttons at that step by 5% each. Conversely, pressing an incorrect button decreases that button's probability by 10% and increases the other two by 5%.

The policy for noxious rooms is similar, but not quite the same. Instead of gradually increasing the probability of jumping after receiving a shock, the probability

immediately jumps to 90% and stays there. However, the noxious stimulus also causes Bob to associate this type of room with pain. Thus, the probability of jumping in all other noxious rooms is increased by 20% whenever Bob receives a shock.

The final policy has to do with movement. Since Bob's ultimate goal is to escape the maze with the lowest possible hunger and pain scores, it made the most sense to combine these two scores into a single reward value that could be used to calculate Q-values for a Q-learning algorithm. The details of this algorithm will be described in a later section, but the resulting Q-values are used to calculate probability values. These probabilities are inversely proportionate to the Q-values, so that a lower Q-value corresponds to a higher probability and vice-versa.

Graphical Representation



R-Matrix

The following table displays all of the possible state-action pairs and their associated rewards:

State	Action	Hunger Value	Pain Value
Locked, appetitive room	Press correct button	-5	-1
Locked, appetitive room	Press incorrect button	+1	-1
Locked, noxious room	Jump (anytime) or do not jump (while floor is not electrified)	+1	-1
Locked, noxious room	Do not jump (while floor is electrified)	+1	+20
Unlocked room	Move to another room	+1	-1
Unlocked room	Move to the exit	-10	-1

Q-Learning Parameters

I provide two parameters that are used in my Q-learning algorithm. The first is the discount factor (gamma), which affects how much the Q-values for all possible next actions are able to alter the reward for the current action. The second is the learning rate, which affects how much the overall difference between the old Q-value and the discounted reward is able to alter the new Q-value. My default discount factor is 50% and my default learning rate is 20%.

Q-Matrix Updating

The Q-matrix is initialized to all zeroes at the beginning of training. Whenever Bob moves to a new room, his current reward values are summed together and recorded. Once Bob either arrives in an unlocked room or unlocks a locked room, his new reward values are summed and the old sum is subtracted from it. This is the R-

value, which is plugged into the Q-learning formula to get the Q-value. Once Bob has completed one episode, the Q-matrix looks like this:

State / Action	0	1	2	3	4	5	6	7	8	9
0	0	0.84	0	0	0	0	0	0	0	0
1	0.28	0	0.16	0	-1.39	0	0	0	0	0
2	0	0.48	0	0	0	0.25	0	0	0	0
3	0	0	0	0	0	0	-5	0	0	0
4	0	-0.28	0	0.6	0	0	0	0	0	0
5	0	0	0.47	0	0	0	0	0	0.38	0
6	0	0	0	0	0	0	0	0.2	0	0
7	0	0	0	0	0	0	0	0	0.02	-1.2
8	0	0	0	0	0	0.42	0	4.8	0	0

Performance vs. Number of Episodes

The first episode took 160 time steps to complete, where a time step is incremented each time a state-action pair is selected (see *R-Matrix* section for a full listing). After the tenth episode, however, Bob was able to find the exit within 27 time steps. This indicates that Bob learned to finish tasks more quickly, as well as find a more efficient route through the maze, over the course of his training.

Changing the number of episodes to five resulted in a predictably worse outcome, as Bob took a total of 149 time steps to reach the exit. Increasing the number of episodes appears to have diminishing returns, though. At the end of a 15-episode trial, Bob was only able to improve his time to 24 time steps.

Case 2: Different Gamma Values

When the discount factor was reduced from 50% to 20%, Bob took twice as long (54 time steps) to escape the maze on the tenth trial. When the discount factor was increased to 70%, however, Bob took more than twelve times as long to find the exit, with a total of 327 time steps on the tenth episode. This indicates that a discount factor closer to 50%, or perhaps slightly lower, is ideal for this model.

Case 3: Different Learning Rates

Decreasing the learning rate from 20% to 10% did not improve performance, as Bob took 139 time steps to exit the maze. However, increasing it yielded negative results as well. A learning rate of 50% resulted in 85 total time steps, and a 70% learning rate resulted in 169 total time steps. This indicates that the default learning rate of 20% is close to ideal for this model.

Case 4: Different Policies

I was unable to run the same model with different policies, but a few different policies are be used simultaneous in the default design. Please see the *Policy* section for more details.

Case 5: Different State/Reward Functions

I was also unable to run the same model with different state-transition or reward functions. However, my model transitions between alternating state-transition functions depending on whether the room is locked or unlocked. It also transitions between alternating reward functions depending on whether the locked room contains an appetitive or noxious task. Again, please see the *Policy* section for more details.