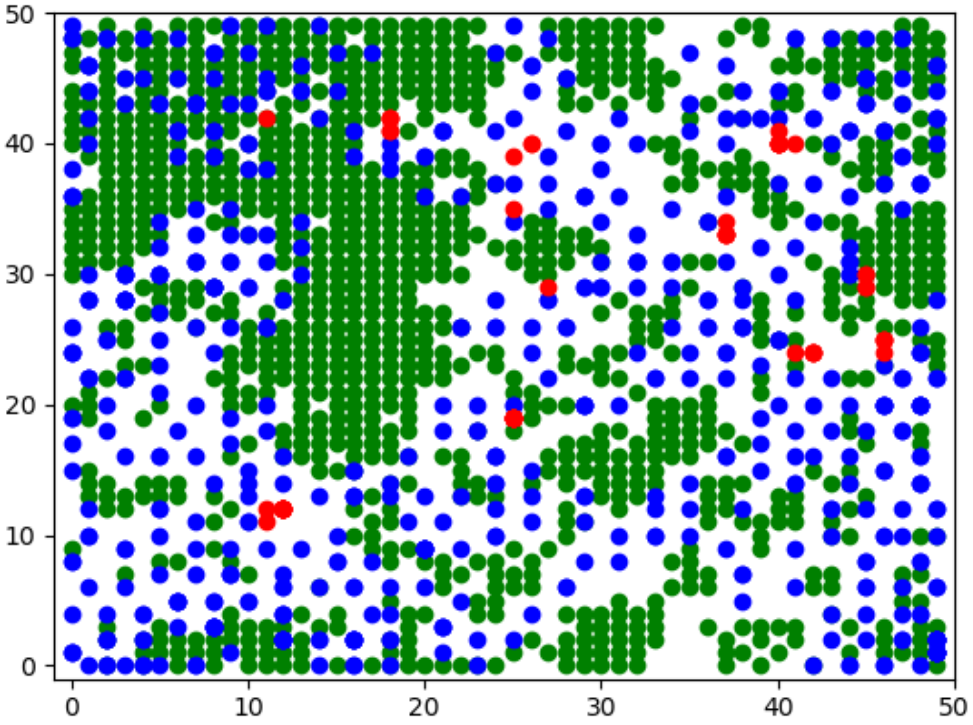
FEBRUARY 5, 2018 / ANDREA8113 / LEAVE A COMMENT

Reinforcement Learning in a Predator Prey model

The code can be found on my github

Predator prey model is a well known branch of the study of the dynamics of biological systems. One of the most frequently description of this model is through the Lotka–Volterra equations, a pair of first-order nonlinear differential equations. The dynamics given by this set of equations can also be derived by a simple agent based simulation in which predators and preys randomly move on a grid, reproducing and dying according to given rules.



But how the overall dynamics changes if the agents do not move randomly? And even more interesting, how does it change during the learning process? M. Olsen and R. Fraczkowski tried to answer to these questions in the paper "An Agent-Based Predator-Prey Model with Reinforcement Learning". We are thus going to take the main idea of this paper, that is to model the brains of each single prey and predator as a neural network in charge to decide which action to make. More in particular, the only decision process is about the movement, as the agent will always eat if possible as well as reproduce. To better simulate the process of learning, throughout the simulation predators and preys will learn which action is best in which situation, through a trial and error process known as reinforcement learning.

The model implemented consists of three types of agents, one which is stationary and two that are able to move on a grid. The two moving agents are the predators and the prey, which also interact with each other to simulate the relationships depicted by the predator-prey model. The stationary agent, the grass, acts as food for the prey. Agents make decisions about their next move based on their perception of the neighborhood and their learned weights. Then the move is evaluated with Q-learning

and the agents are given a reward. Next the agent will eat if the new position contains the predators or the preys respective food agents, and they might reproduce. An update of the model consists of all agents moving, which is done in a random order.

Stationary agent: grass

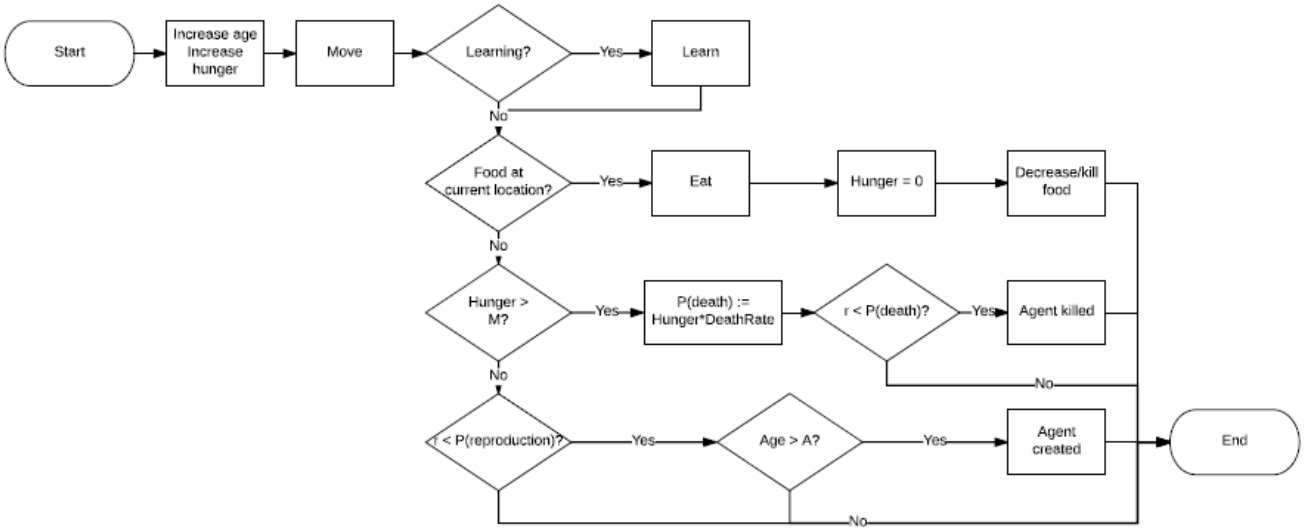
The grass agent is the most simple of the three agents used in the model, due to it being stationary and not having to learn. When a grass agent is created it's food count is set to 1. The food count is then decreased by a set value C_food every time a prey is consuming it. Once the food count reaches 0 the grass agent is considered dead and removed from the system. The stationary agents are also able to reproduce like the moving agents. For each time step t a grass agent may reproduce with the probability P_Reproduce. Unlike the moving agents there can only be one grass agent at a location at any time, and hence a grass agent may only reproduce if there is a location in it's Moore neighbourhood which does not contain a grass agent. If this is the case, one of these locations will be chosen at random and a new grass agent with food count 1 will be added to this position. It is important to note the usage of grass agents in this model. Many classical population dynamics models do not take into respect the fact that there might be a finite amount of food available to the prey. By adding the grass agent to the system it is also possible to study the case of 'overeating', i.e. the case in which the prey population grows too large causing the food to become so scarce that the entire prey population dies from starvation. It is also interesting to study the clustering effect on the grass agents that occur as the prey population grows, as this clustering effect also is observable in real ecological systems.

Moving agents: predator and prey

The predators and the prey agents have very similar behavior and therefore both are modeled in a similar way. A predator or prey consists of an object with several parameters depending on its species as well as two individual parameters and twelve

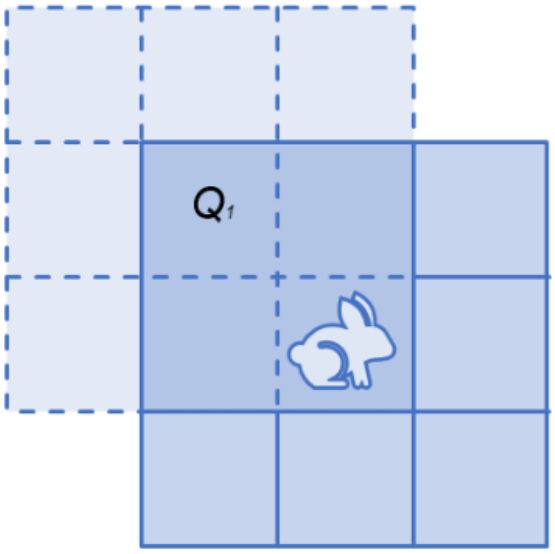
individual learning weights. The main parameters of the agents are its age, which is updated time step, and its hunger. The hunger parameter grows for each time step the agent doesn't find food, and is set to zero when it eats. Other parameters of importance are the minimum reproduction age A and the hunger threshold M, which dictates the minimum age and hunger for the agents to be able to reproduce and die of starvation. Lastly there are also parameters regulating the probability of death and reproduction.

For each update an agent performs the actions in the flowchart in the figure below. The chain of actions starts with an update of the age and hunger parameters of the agent. The agent then proceeds to move on the grid. When moving the agent have a probability of making a random move. The reason for this is to allow the agent some exploration, from which it can learn new behavior. If the movement is not made randomly the agent will evaluate its neighborhood in order to make the move it believes is the most beneficial. This is described in more detail later. The model is implemented in such way that agents only learn for the first 500 time steps. Thus, if the time step is lower than 500 the agent will evaluate the move it made and learn from it. If the new position of the agent also contains a suitable food agent, i.e grass for the prey and prey for the predator, the agent will proceed to eating. This decreases the hunger of the agent to 0 as well as decreasing or killing the food agent. If no food is available and the hunger of the agent is lower than M it is at risk of dying of starvation, with the probability P_death = hunger*Death_rate. If the agent survives it has a probability P_reproduce to reproduce. The reproduction in this model is asexual, and occurs only when the age of the agent is bigger than A, and the agents hunger is smaller than M. If the conditions are fulfilled a new agent is created, which inherits the weights from the parent agent.



The agent's 'brain':

As said earlier, a key difference between this model and other predator-prey models is that the species are not moving randomly. Instead they make decisions based on their perception of the neighborhood, which is also used as basis for the learning. An agent evaluate the grid locations in its Moore- neighborhood. For each location 12 features are calculated, and its Q-value is obtained by computing the dot product between the features and the weights of the agent.



The agent will then move to the location with the maximum Q-value. The features are: the ratio of opposite agents in neighborhood and in the world, the ratio of same-type agents in neighborhood and in the entire world, the ratio of grass agents in

neighborhood and in the entire world, the ratios of opposite agents for each grid location compared to the neighborhood.

Learning

Reinforcement learning is a branch of machine learning in which the agent tries to apprehend the ideal behaviour in a given contest, through a reward-based trial and error approach. In other words its aim is to reach an optimal action-selection policy. It does not belong to either Supervised or Unsupervised learning areas, since there is not a ground truth supervising the process, instead a notion of reward has been given in order to guide the learning. Compared to the latter areas, one of the biggest Reinforcement learning strengths is the ability to operate model free, that is it does not require any previous external knowledge of the environment in which it operates (other than the reward function).

Q-learning is one of the most used Reinforcement Learning techniques, thanks to its simplicity and its ability to find the optimal action-selection policy of any Markov decision process.

The general functioning is as it follows:

•The problem is time discrete and for each step is associated with a state s_t.

• At each time step the learning agent performs one action and, based on the current and previous state, it receives a reward, usually quantified as a numerical value.

• The aim of the agent is to maximize the total reward.

• The action decision is based on the function Q(a,s_t) that quantify the 'Quality' of the action a given the current state s_t.

• After each action the Q-function gets updated according to the following equation:

$$Q(s_t, a_t) = (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max_a Q(s_{t+1}, a))$$

where α is the learning rate, and γ is the discount factor, a constant that controls the importance of the future step reward. In the implemented model a linear value function Q-learning is used. The Q-function is computed through an artificial neural network without hidden layers:

$$Q = \sum_i w_i \cdot f_i$$

where *f* are different features extracted by the agent from the environment and w are the weights of the network. The main advantage of this approach is that the Q-value for several different states can be expressed using only a few parameters, i.e. the weights of the network. Adding one or more hidden layer would have increase the ability of a well trained network to perform better, however way more iterations would have been necessary for the neural network to be trained. Furthermore, with the proposed topology each weight has an understandable meaning: it determines how much the associated feature is important; the bigger w the more (positively) important *f* is. The learning process can thus be expressed as the 'tuning' of the weights in order to get an optimal action-selection policy:

$$w_i = w_i + \alpha \cdot (r + \gamma \cdot max_{a'} Q(s', a') - Q(s, a)) \cdot f_i$$

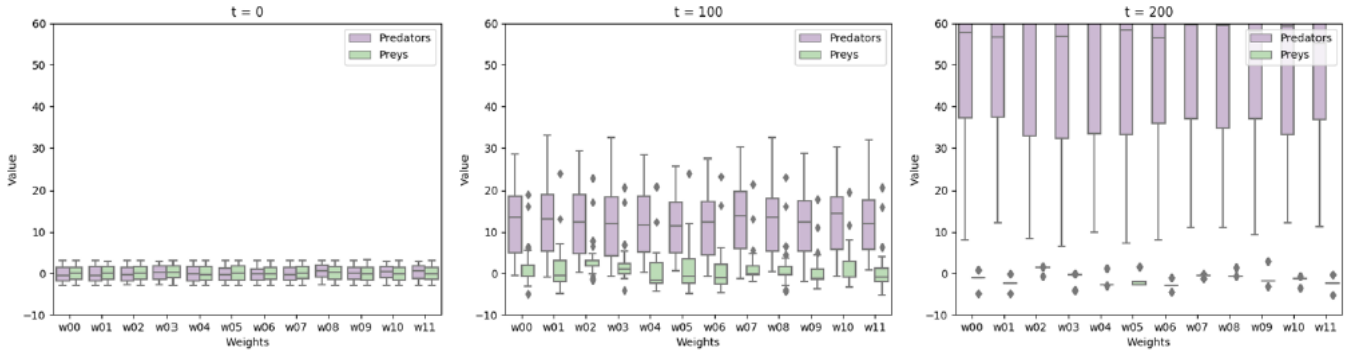
where s is the state before the action a and s' is the state right after the action. If the reward plus the future gain is bigger than the actual Q value, the weight will increase and vice versa. Furthermore, there are no limits hence the weights are free to assume both positive and negative values. The parameter α as mentioned before controls the change of the weight. At the beginning of the simulation it has a value of 0.05 which decreases linearly during the iterations, reaching zero at iteration number 500. The discount factor γ has been set to one and remains fixed throughout the whole learning process. The reward function for preys and predators is the same and has been set as following>

$$r = 2S \cdot t + O \cdot t$$

where S is 1 if the agent moves on a cell where there is an opponent and zero otherwise, t is 1 for the predators and -1 for the preys. In this way, the reward is positive for predators and negative for prey.

The results: ***have the agents really learned?***

The first and more interesting question to ask ourselves is if the predators and the preys have actually learned. As previously mentioned, one of the advantage of a single layer network is that we can easily understand the weights meaning. The weights w0 w1 w2 denotes the number of agents of the opposite type, same type, and grass type in the proposed cell neighborhood respectively, whereas the last 9 weights represent the ratio of opposite type agents in each Moore neighborhood cell. We would expect the prey to get globally negative weights, which means they would learn to avoid predators, reducing the score of 'dangerous' cells. Similarly, we would expect predators to obtain globally positive weights, to motivate the agents to get closer to the prey.

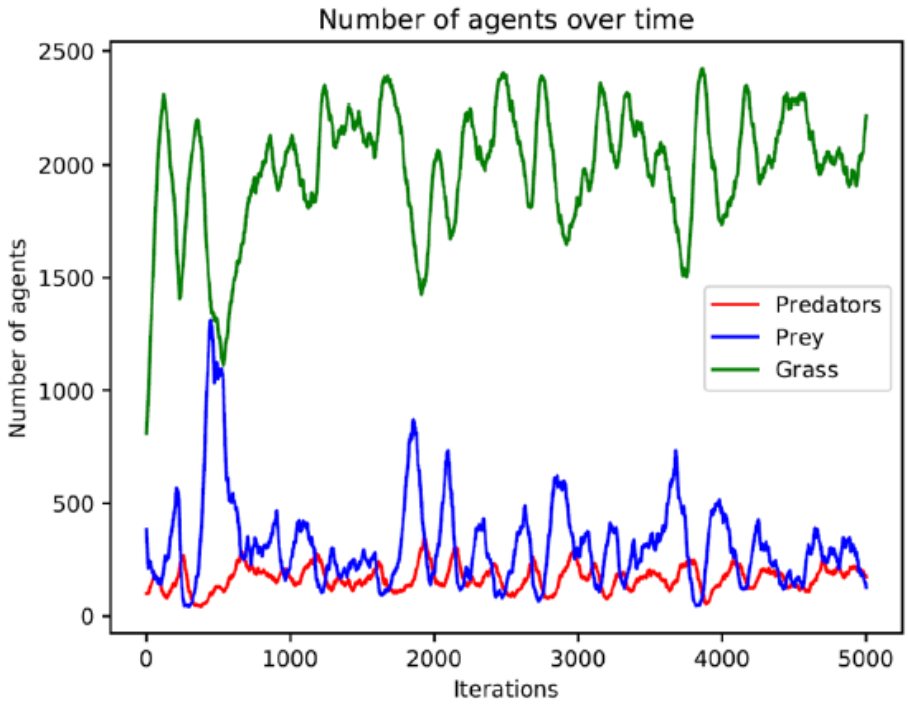


In fact, during the training phase this is what happens. It can be seen in the box plot that the weights of the predators increase quickly whilst the weights of the prey tend to fluctuate before finding a global consensus for all agents of that type. The weights of the prey are generally low or negative, except w2 which represents the affinity for grass. Note that weight w1 is weakly negative and it represents the affinity with agent from the same species. The prey will try thus to avoid other prey agents, since they would be more likely to meet a predator if they do. On the contrary, predators in general tend to be attracted by their similar, resulting in herd behaviors. We can also note that the predators are attracted to grass (w3) as they may have associated the grass with the likelihood to run into a prey. The stability of the final-state learning of the prey can be explained by the selective pressure that has been applied on that species. Since they share their exact genetic traits with their offspring, the

final-state learning is an effect of both an evolution (learning) and a selection (fitness, survival and reproduction of an individual with a given weights set).

The results: ***can this model be described by Lotka-Volterra equations?***

It is important to notice that there are several stochastic component in the model, such as the initialization position of the agents, the early movements, the deaths and reproduction. This, added to the chaotic nature of the simulation ensure that even running the simulation with the same parameters will give sometimes strongly different results. Anyway, typical result look like this:



Overall, the trend shows constant fluctuations for all the species with different magnitudes. Similarly to the Lotka-Volterra equations, there is an inverse correlation between predator and prey (as well as between prey and grass), since when there are few predators then the number of preys will increase, this will comport a consequent increase of predators that will reduce the number of preys. The predator numbers will drop again due to the scarcity of preys and this cycle continues.

Can we thus say that the model resembles the Lotka-Volterra model?

My personal opinion is that no it doesn't, for several reasons. The first one is that the model in this case is a time delayed one; in other words, the current state of the

system does not only depend by the previous one, as the effect of the hunger does not show up immediately. In fact, time delays provoke oscillations in the population dynamics. Another (bigger) cause of these oscillations are, as previously said, the stochastic components. Taking a Malthus model, the standard deviation is given by:

$$\frac{std(N(t))}{< N(t) >} \sim N_0^{-\frac{1}{2}}$$

Although this is not the case, this formula gives us an idea of how, since our population size is small, the oscillations are actually strong and cannot be omitted. This reveals another important limit of our simulation, that is the restricted size of the simulation, which enhance also the difference between a discrete and a continuous dynamics.
Finally, while the Lotka-Volterra assumes the agents to be uniformly spread, in our model we observe how all the three species (for different reasons) tend to form clusters.

Published by andrea8113

View all posts by andrea8113

Uncategorized

Leave a Reply

Enter your comment here...

PREVIOUS

Continuous Attractor Neural Network

NEXT

Anomalous random walks and their Mean First Passage Time

SEARCH

Search …