CSE 240: Assignment 5

Introduction

In this assignment you will be implementing an AI to play the Snake game using a simple reinforcement learning algorithm: `q-learning`. This assignment is supposed to be written in python3. You will run this assignment using the command `python3 game.py`.

1 Snake: The game

Snake is a common type of video game where the concept is the player maneuvers a line which grows in length with itself becoming the primary obstacle. There are hundreds of versions of this game with slightly to completely different rules.

1.1 Rules

For this assignment you will stick to the following rules:

1. Snake starts at size 1.
2. There is one source of food at each instant.
3. The game ends if the snake eats itself.
4. The game ends if the snake hits a wall.
5. The game ends if the snake starves.

2 Implementation and objective

For this assignment a lot of the work has been done for you. You are given four python files as starter code. The contents of these files will be described in the next section. You are required to submit all 4 of these files to Canvas.

These files are:

1. `helper.py`: A file containing helper variables, and functions that have uses in different places in the code.
2. `snake_agent.py`: A file that contains the reinforcement learning agent and its associated class and functions.
3. `board.py`: A file containing the classes that handle the current state of the board as well as the functionality of the snake.
4. `game.py`: A file that contains the `__main__`and essentially acts as the driver for the rest of the code.

2.1 Objective

Your objective in this assignment is to implement the `q-learning` algorithm in order to make an AI agent that can play the snake game with the rules mentioned above. There is not a lot of code to write as most of this functionality is already written. Essentially you need to modify the code in a grand total of **four** places where `#YOUR CODE HERE` is mentioned. We will now go over the code in each file to explain what needs to be done.

3 CODE

As mentioned above the code is divided into 4 files. We will go over each file and its contents and purpose here.

3.1 helper.py

This file contains variables that are used in multiple places in the program. There are too many to mention each one here, but the names are self-explanatory. This file also contains the name of the file that will store the `q-table` that will be generated by the **training** step. This file is called `model.npy`.

This file also contains a function `save` to save the `model.npy` file and `load` to read the saved `model.npy` file. To assist in this process there is also a function `np_error_checker` which checks the format of the contents being saved and loaded.

There is also a function called `make_args` which is used to create arguments to pass to this program with default values already defined in this file. To change how the program works the easiest way is to change the values mentioned in this file.

All the functions in this file have been completed and no changes are necessary except for when changing number of iterations for training, testing etc.

3.2 snake_agent.py

This is the file that has the class and functions necessary to implement the snake game. The class `SnakeAgent` has the necessary variables to implement the functions.

`__init__`is the constructor for this class, it sets the initial values for the actions (moves made by the snake), `Ne`, `LPC`, and `gamma` variables which all help when calculating the learning rate and deciding the next action. if you're unfamiliar with how to modify the learning rate based on the variable `LPC`, start out using the value `0.7` for lr (learning rate) and then once you get your code to work, work on this part. `gamma` is used to balance the immediate and future reward.

The `self.Q` variable is a numpy.array variable which is the q-table. The N variable can be used to see how making a move will affect the next state, only Q is used to save and load for calculating moves.

There are multiple functions defined in this class. `set_train` is used to set the `_train` variable to `True` to indicate that we are currently in training mode. Conversely, the `set_eval` function sets the `_train` variable to false to indicate that we are in testing or evaluation mode.

the `save_model` function is used to save the q-table model developed during training and the `load_model` is used to load the saved q-table model into `self.Q`. the `reset` function is used to reset the points to 0 and actions to None.

There are two functions you need to write in this file. the `helper_func` will be used to get the possible moves that can be made given a game state, where you can get the different values like current position of the snake's head, current position of the food, where the snake's body is, which direction is the food compared to the body and so on. All this information can be returned as a single variable `state`.

the other function that needs to be defined is the `agent_action` function which is where the q-table model will be built when in the **training** mode and when in **testing** mode the best action to be made will be calculated and returned. Note that in order to build the q-table you still need to calculate what the best action that can be made is. Use the `compute_reward` function to determine the best action. **This function must return only one thing, the action to be performed.** The action is `0 or 1 or 2 or 3` indicating up, down, left, and right respectively.

3.3 board.py

This file has two classes implementing the functionality of the board and snake respectively.

3.3.1 class BoardEnv

Class `BoardEnv` defines the board environment. Its constructor initializes the board and food positions tothe default ones indicated in the `helper.py`.

Most of the functions in this class just call their corresponding implementations in the `Snake class`. These types of functions are indicated by using a "*" in the comment above them.

This class does contain the `draw` function which draws the different parts like the board, snake, and food. It uses the `pygame` class to make this job easier.

This class also contains the `display` function which calls the `draw` function. This function also initiates the pygame module.

3.3.2 class Snake

the class `Snake` handles the functionally for the snake. The constructor initializes the default starting position for the food and the snake. It also sets a number of steps after which the snake dies if it hasn't consumed food by then.

the `reset` is used to reset all these values in the case where the snake dies, and we need to reset tostart again.

the `get_points` function returns the current number of points the snake has in the game.

the `get_actions` function returns the different directions the snake can move

the `get_state` function returns the current state of the snake, it's head position, body size and current food position.

the `move` function takes in an action as a parameter and performs the move, moving the snake in that direction, and then creating a new food location for the snake. This function also returns if the move made killed the snake by returning `True` (based on the rules mentioned above). If not, it returns `False`. The `step` function calls the `move` function and returns a number of things including the current state, points and if the snake is alive or not.

When the snake eats its food, the `handle_eatfood` function increases the points the snake has and then resets the steps since last food was consumed. It also calls the `random_food` function which creates a new food location randomly. This is done with the help of the `check_food_on_snake` function which makes sure the newly created food location isn't on the snake.

All the functions in this file have been completed and no changes are necessary.

3.4 game.py

This file is the driver file for the code in all other files. This file contains the class `SnakeGame`. This class initializes the default board environment `self.env` and the default snake state `self.agent`.

The `play` function is the main driver. It first runs the training code as many times as indicated in the `helper.py`. Then it runs the testing code. Then it runs the instance of the game where it displays the snake game.

The `do_training` function is called when the program is in training phase. The variable `NUM_TO_STAT` will give average statistics every N number of times the game is played, by default this is set to 100. You are to write the code in the place of `#YOUR CODE HERE` in this function that will call the `self.agent.agent_action` function in training mode and update the `self.agent.Q` variable. The generated q-table is saved by the `self.agent.save_model` which is already called in the correct place. Note that there is a block of code commented out which will print how well your training is proceeding.

Similarly, in the `do_testing` function you are to write your code to call the `self.agent.agent_action` function in place of the `#YOUR CODE HERE`. Again, there is a block of code commented which will let you know how well the agent is doing when in testing mode. Loading the saved q-table model is already done using the code `self.agent.load_model`.

The next function is the `show_game` function which will display `NUM_DISP_ITER` number of times. Right now, `NUM_DISP_ITER` is set to `1` and since the `agent_action` function is not implemented fully, it doesn'tshow the snake moving. As you write your code this will change.

The last function in this file is the `__main__`. Here we just parse the arguments mentioned in `helper.py` and pass it to the `SnakeGame` class.

4 Summary

In this assignment you are to implement an AI that can play the snake game using q-learning. You are to submit four files as mentioned above. There are only four places where you need to make any changes for this assignment which are

- helper_func - writing helper code for agent_action function
- agent_action - writing the learning and inference for the q-table
- do_training - calling the agent_action function appropriately
- do_testing - calling the agent_action function appropriately.