Honglu Xu
EECS 513
Final Projest

Apply Artificial Neural Networks on AIs to Play Video Games. ¶

Same part with EECS 491:

Note: This is a project that cannot all implemented in the Jupyter Notebook, and it is too hard to separate the code and copy them into the notebook, so there will not be any code showing in this notebook. All the file-names and descriptions are listed below, and I will try to explain the code part as clear as possible.
Please make sure you have correctly the configured your envirionment before you run the code. The main packages that will be needed are: "pygame", "numpy", "pgmpy", "keras".

Files:

1. Snake.py: Main game file. Contains all the modules needed to run the game.
2. VisionNN.py: Vision part of nerual networks. AI model for get input data from the screen pixels.
3. InputAgent.py: Auto input agent for naive logic game input generations. Will be used to train the logic AI.
4. LogicNN.py: Logic part of nerual networks. Will be used to generate output (player input) through the ANN.

About the game

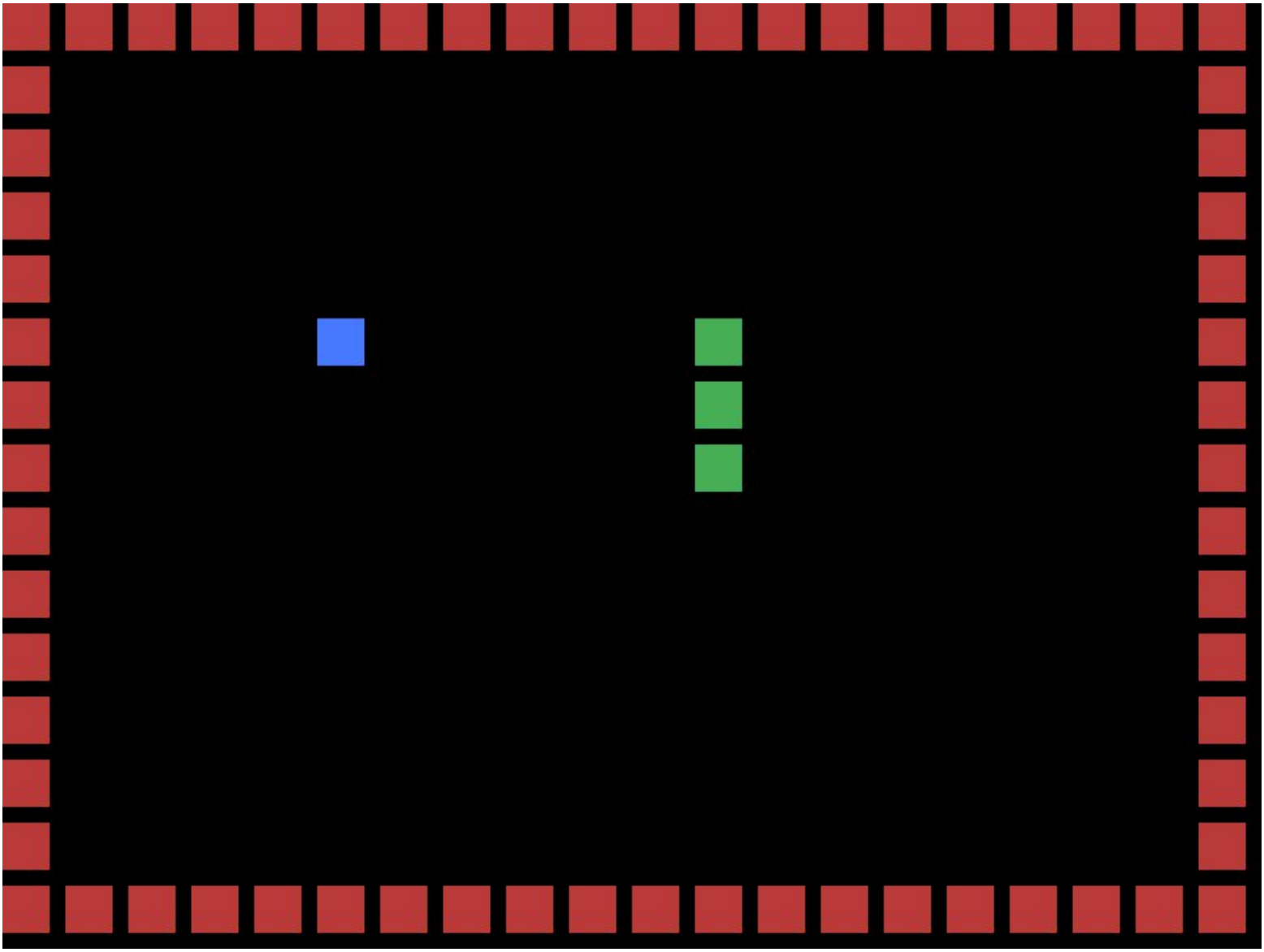
To build a game video AI, we firstly need to build a game. In this project, I will use the package "pygame" as the game engine.
The main game logic is simple. As long as we are running the game, it will:

1. Check all the input I/O from the player/system.
2. Update the game, which will apply all the inputs on the game and calculate all the data for next frame.
3. Render the next frame.
4. Keep looping from step 1 until end.

The game I will use for my experiment is "Snake". I chooese it for several reasons.

1. It is easy to implement.
   The snake is sort of an simple game. We might have encountered this game many times, we are familiar with it, and since it is in 2D and block based, we do not have many rendering part.
2. Game logic is simple enough for a simple AI.
   The game logic for snake is simple, for the player part, the input should be in 5 statement: "up key", "down key", "left key", "right key" and the "r key" for restart the game. And the output (feedback) should also be simple: survived (continue the game), ate the apple, and died. Thus, we have 5 input and 3 feedback statement, which should be simple enough for a starter experment.
3. Game logic can also be complicated.
   The game can also be complicated if we want to behave very good on this game. Sometimes we surrounded ourselves as a circle, then that area is a "trap", we should avoid our snake goes in. Sometimes we need to go through a narrow path that seems dangerours. Sometimes we need to count the steps so we can survive just one step early. Thus, it also can be a complicated game.

This is the original game scree capture. I used the code from the tutorial of Python official site [1].



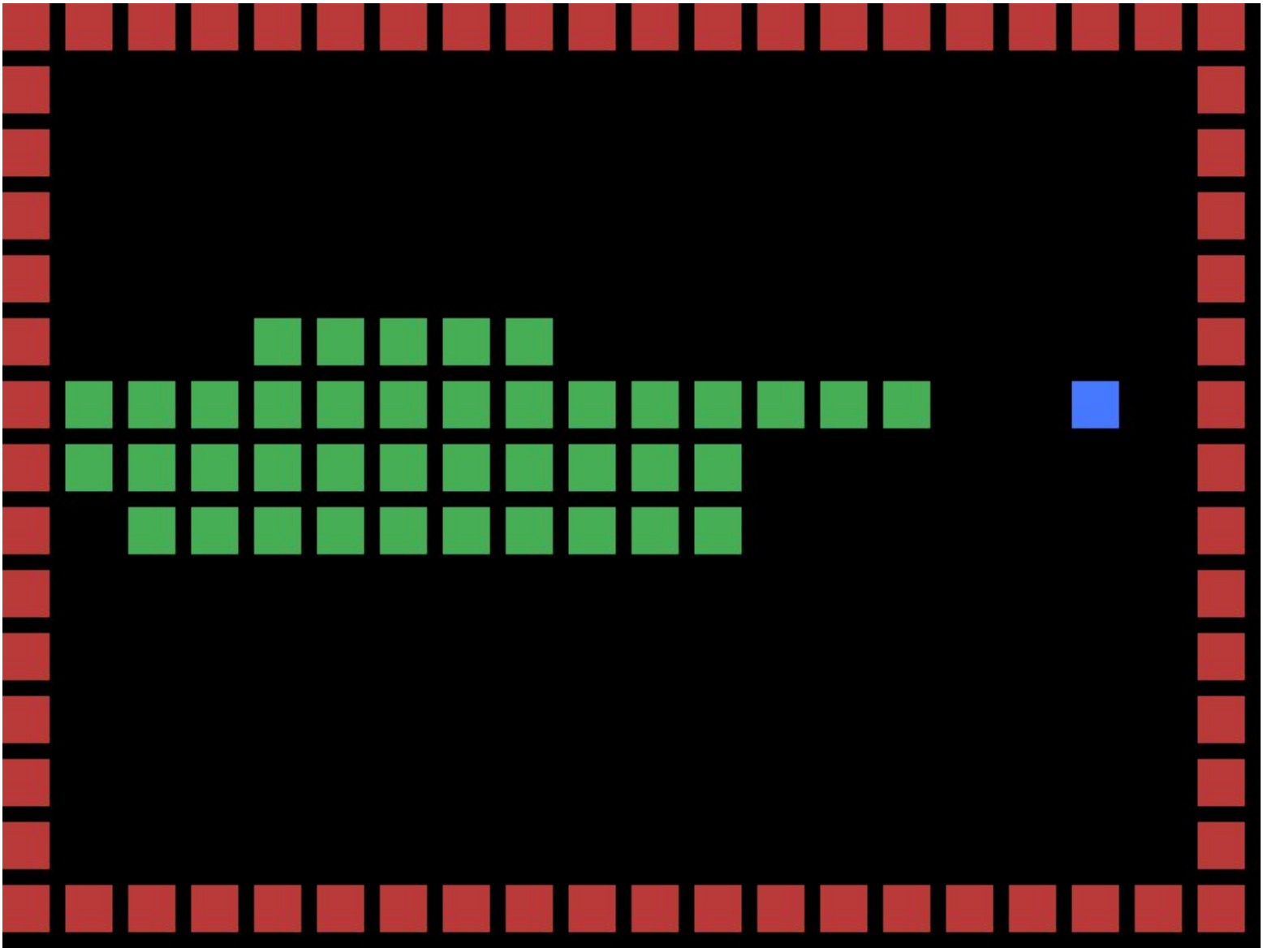
Game Tags:
If you want to try the code, you might need to configure the game tags that in "Snake.py" so that it will work as you wanted. Here is the list and descriptions of the game tags.

1. using_AI: Main tag that will enable or disable all the logic AI.
2. AI_type: Logic part AI type.
   random: Randomly generating inputs.
   bys1: Decision trees with believe network.
   beta: Alpha-beta algorithms.
   bys1nbeta: using both bys1 and beta
   NN1: Neural network model1.
   NN2: Neural network model2.
3. beta_step: Step configuration for alpha-beta.
4. train_Model: Logic models that will be used for taining.
5. critical_train: Train logic models only in the critical condition.
6. reading_Model: Reading logic models from file (file name hardcoded).
7. test_Model: Logic models that will be used for testing.
8. using_VS: Main tag that will enable or disable all the vision AI.
9. Vision_type: Vision part AI type.
   NN1: Neural network model1.
10. train_VModel: Vision models that will be used for taining.
11. critical_vtrain: Train vision models only in the critical condition.
12. reading_VModel: Reading vision models from file (file name hardcoded).
13. test_VModel: Vision models that will be used for testing.

For default, you can disable all the tags, then enable it one by one to see if it works.

For EECS 531

Problem encountered: Unlike human brains, which will be continuously influenced by new input data and time based old input data (which is the memory), simple AI may only get the data of only one picture (single time slice) and need to generate the information that may based on time.
We can have a simple example to illustrate this probelm.
In the demo picture above, where is the snake facing to?
Judged by the shape, it could not face left and right. But for a single time slice, we could not tell whether it is going up and down, so does the AI. Another example is showing below.

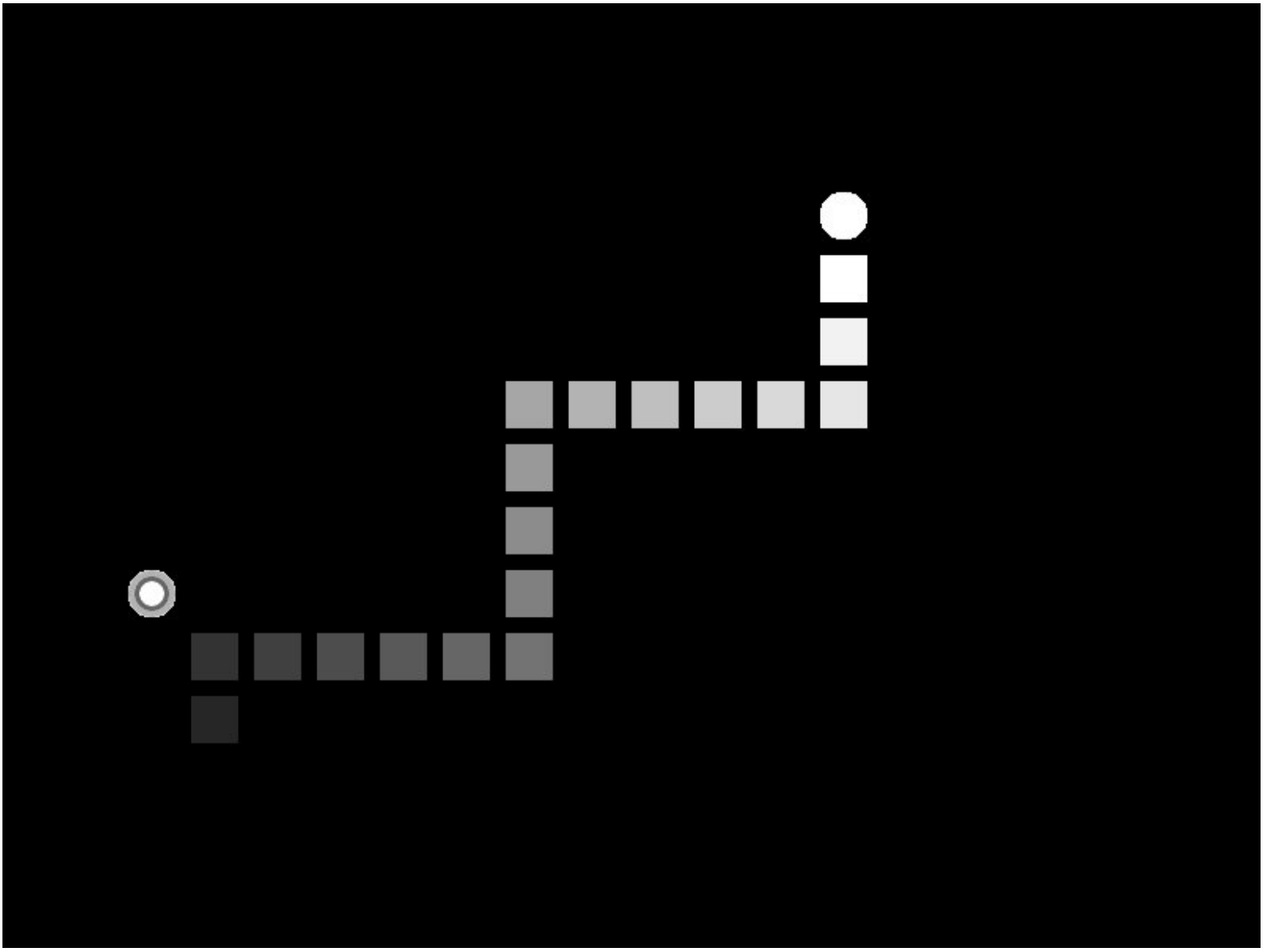


For the example above, even we know the snake's head is the right most green block, we still can't get enough information for the snake tails sequence picture from the. We know that this will happen even on a time based human brain, sometimes we can't tell the squence of the tails when we play this game, but for a person with good logic and memory, he can tell the squence of the tails for most of the time.

Possible solution:
These are all planned solutions for this, unfortunatly, I ran out of time for this project. So I won't do any of these listed solutions for now, I will only use the simplest solutions for this problem.

1. Tune the logic AI so it can be based on time and memory.
   This should be the ultimate solition for this, and it is also simulating the human behavior. Mostly, the human eye part is only doing the "input" part, which is capturing the image and pass it to the brian, and it is the brain's logical part to tell what is the current statment and how to play the game. To achieve this, RNN based networks may be needed, and as long as memory module. The cons for this solution is very hard to implement, and will cost a lot resources for training.
2. Motion detection for CV.
   We can also use the motion detection part for this problem, which is also based on the time slices. But, we have cons for this.The snake's move is based on the blocks, which is not really continuously. The tails will also replace to each other, which is not good for motion detection.

The simplest solution: change our game so it can be easily detected, and do not need to be based on time squence.
After serveral trys, I decided to start from the simplest method first, so I can have a smooth progress for later. The simplest way is to change the color for the game, so eveything can be detected within only one frame. It will surely not be the solution for other games, but for this test state, we can do it as a testment. The final hehavior is showing below.



As you can see. I changed the colors for all components. To separate the head from the tail, which we will need to tell the direction, I cahnged the shape from box to circle. The tails are in gradient colors now so we can tell the squence of each block. I marked the food, so it can be separated from everything else. All the wall is canceled, so w can reduce the output. To reduce the input, I changed all color from RGB to grayscale.

Now, let's do the vision part. The object of the vision part is to get information as much as possibile from the pixels we get from the screen. For a easy start, we could output the coordinates for all the components so we can farther use it in the logic part.

1. Manully feature detection

This is the simplest way to use to detect the feature, which is from the first class. We can use feature detection manually.

1. Use the self-defined feature go through all the patches we have.
2. Find the maximum relation ship for the certain feature
3. Record the patch posision
4. Translate the posision to the game coordinates.

Since we defined the components of the game by ourselves, we can get he feature easily. By using this method, we can fastly get the position for player head and the food. However, the cons are, it is not a genetic way for the solution. Like all the tail blocks in gradient colors, we cannot self-define infinity numbers of features for detection. Also won't work if we can define the features precisely.

2. Naive approach of ANN

This method is implemented as the name of "NN1" for vision part

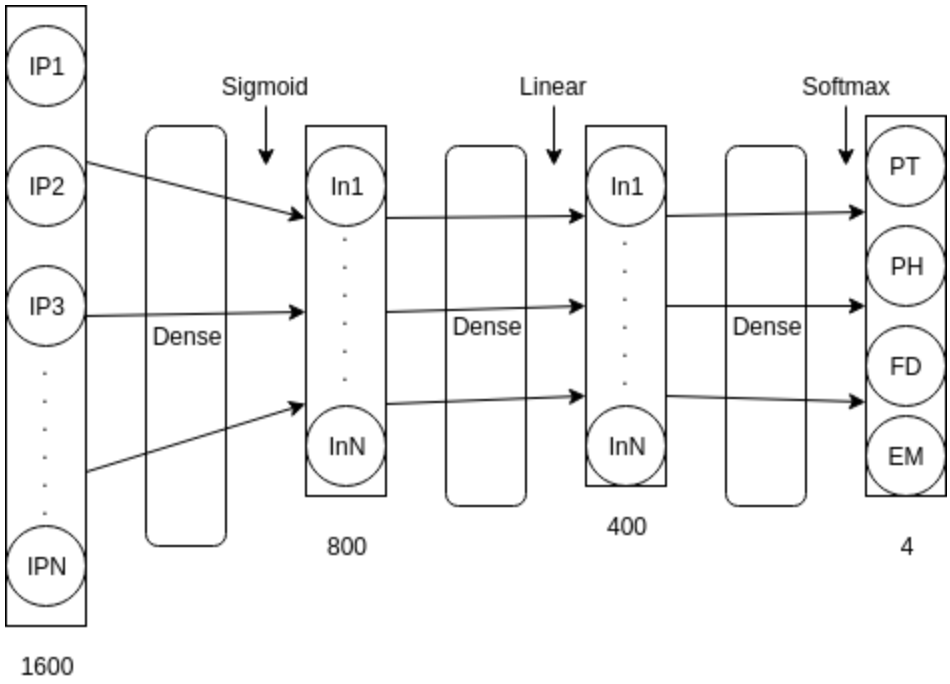
Since we can't define all the features by ourselves, other methods will be needed. We can use PCA and probabilites as the classification. But for the pixel styled input data and labled output data, neural networks could be a great solution.

For a naive approach, we are not doing the whole image trainning at a time. Since the snake game is block based, we could separate the screen to blocks so we can do the classfication for all the blocks. The blocks show below.

We see that the whole feature are contained in a single block, so we cound do the classification. Thus the whole process will be:

1. Separate the whole screen into blocks and record the position for each block.
2. Use the neural networks for classification.
3. If the output is one of the blocks above, we record this block.
4. Retrieve all the usful blocks game coordinates.

Since the feature is actually small, which is 40 by 40 pixels, and at this time, the feature won't change at all, and we only have 3 features, so we could use the fully connected networks only for this situation.



As showing above, we have 4 layers for this network, and 2 of them are inner layers. The input layer is passing the pixels as input, which we have 40*40, we used 1 activation function "Sigmoid" and another as linear. The output is the onehot labeled. "PT" is "Player Tail", "PH" is "Player Head", "FD" is "Food" and "EM" is "Empty". With this classification network, with only several steps of training we can easily do the classification for 98%.

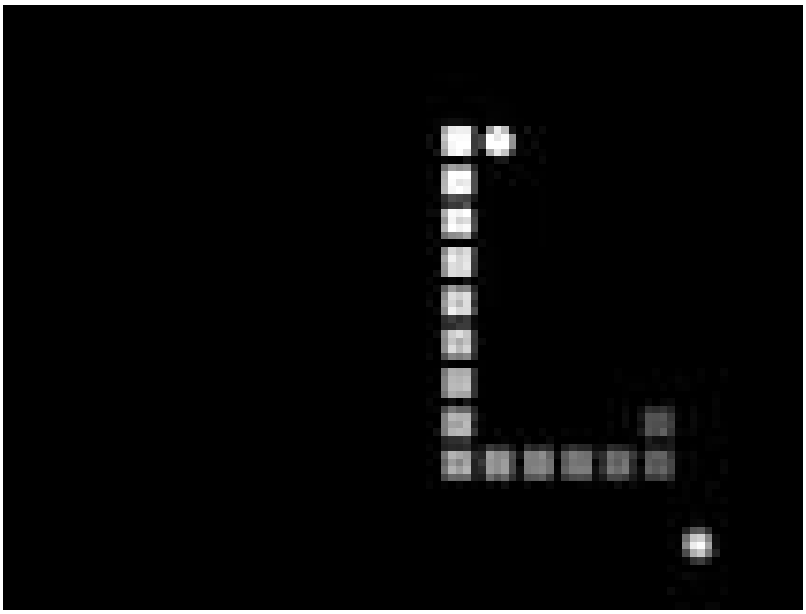
3. Genetic approach of ANN

This model will be presented as the name of "NN2" for the vision part.

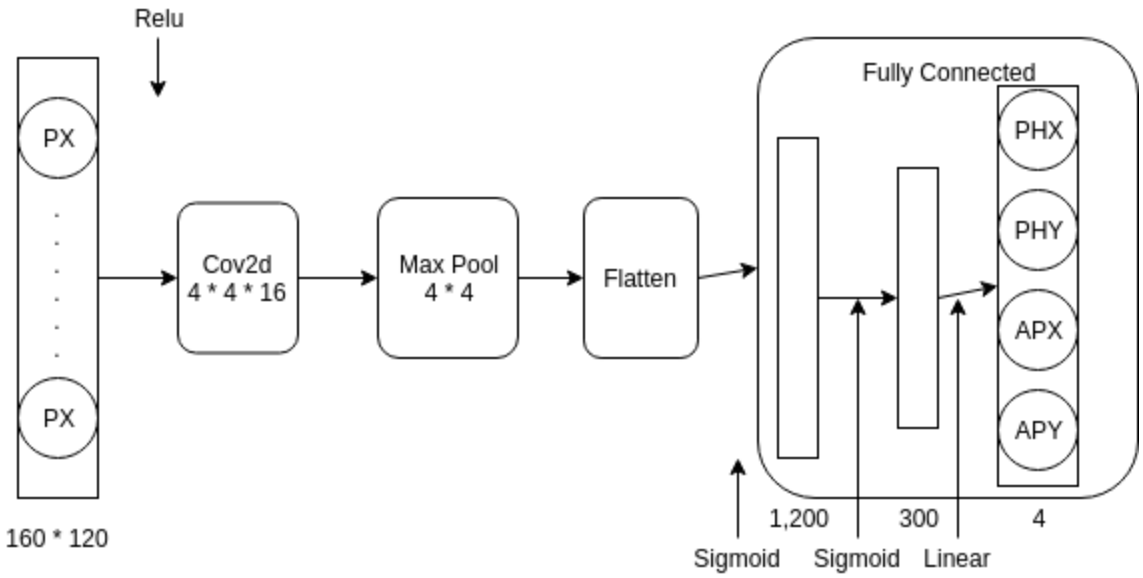
The method above is based on all the internal data that we already know for this game. Since we build the game ourselves, we can use that method. However, for most situation, we cannot access the internal data, or it is too tedious to generate all the coditions from the game. Thus, a genetic way should be used if we want to apply this method on other games.

One method I came up with is use the whole sceen pixels at the input data, not the blocks. We can capture the whole screen pixels at the input data, and give the feedbacks that we want to train the neural networks, then, it will probably learn the thing we wanted it to learn,

To deal with the input data, 800 by 600 pixels is obviously too many. Thus, I used the "thumbnail" function in the PIL package to shrink the image to 160*120, which should be small enough for trainning.



Above is the 160 by 120 image that I generated. As we see, it blurred much. Though the player head and the apple are similar in shape now, but there should be enough details that the neural networks can tell, so the blurred image should be fine. To start, we should just test the player head and the apple's coordinates to see whether it will converge.



Above is the neural network I built for this method. Since we will need automated feature detection in a large image this time. convolution network will be needed. Then the max pool layer will extract the most activated area for each feature. We flatten the 2D layer to the 1D layer, and we can use the fully connected layer and feedbacks to train the network. The output will be: "PHX":player head x; "PHY" player head y; "APX": apple x; "APY" apple y. Since the features for our game are simple, we don't need many features for the Cov2d part, 16 should be enough. And since the snake is block based, all the out put should be game coordinate, which ranged from (0,20) for x and (0,15) for y (20 blocks for x, 15 blocks for y totally). Lastly, since it is not doing the classification this time, we should change the loss function to linear. I used "mean_squared_error" for this test, but there should be a better way to define the loss function.

By theory, after many trails, the accuracy should rise to a decent number. However, as I did the experiments several times, the network seem not converging. I am not sure whether I did some wrongly, or this network is just not working for this situation. For some steps, the predicted outputs did got very precised values. But there also exists some steps that the outputs are far from the correct numbers, no matter how long I trained this network.

The conclution is we still need more works to build an AI that can be geneticly fit in many differnet video games.

Reference

[1] Python Official Site, *Snake with Pygame*, https://pythonspot.com/snake-with-pygame/ (https://pythonspot.com/snake-with-pygame/)