Final Project Report

Osvaldo Moreno and Paris Hom

Team Strawberry Cake

CS 179M

Dr. Christian Shelton

August 28, 2020

Design

Offensive Agent

The Offensive Agent uses a Neural Network to compute the optimal action given the current state. The Neural Network takes in four features: distance to the nearest enemy, distance to nearest food, distance to our side, and the current score of the game. These four features are features that we thought were useful in evaluating the current game state. Given these features, the Neural Network outputs a single value representing the value of the game state.

The decision to use these four features are as follows:

- distance to enemy
    - Offensive Agent should know how far or close the enemies are.
    - help the Offensive Agent try to avoid going to places where the enemy is near and try to survive longer.
    - prevents the Offensive agent from wasting time going back to the opponent's side of the board.
- distance to food
    - main goal of the Offensive Agent is to increase the team's points to win the game
    - To get food, the Offensive Agent needs to know the distance to food.
- distance to our team's side
    - to increase our team's points, the Offensive Agent has to actually bring the food back to our team's side
    - this feature will help the Agent figure out what actions will help it get back home (by reducing the distance to our team's side)
- current score of the game
    - tells the Offensive Agent whether the team is winning, losing, or at a tie.
    - help the Offensive Agent decide whether it should go back home more food or less food.
        - For example, if the game is currently at a tie, then it suffices to bring back one food pellet to start winning the game.

There is also a policy implemented so that once 20% of pellets have been eaten, the offensive agent returns to the team’s side to increase the team’s score. This is to ensure that our team will always win by retrieving slightly more food than the enemy.

Neural Network

The Neural Network is used by the Offensive Agent to evaluate actions. The Neural Network takes in the 4 features defined above. The Offensive Agent generates these features for every action it can take at the current game state and passes it to the model. After evaluating the four features, the model produces one output, which is used to see the value of those features (and gameState). In order to optimize the actions the Offensive Agent takes, the feature action pair that returns the highest Q value is chosen each action.

The data for the model is acquired by running Agent0 10 times for each color on defaultCapture, bloxCapture, jumboCapture, and fastCapture (160 games total). After each game is completed, the features and outputs of each decision the Agent0 Offensive Agent made are recorded into the trainingInput.pickle and trainingOutput.pickle files, respectively.

We gathered training data from Agent0's Offensive Agent because it worked well against the baseline team. This also helped speed up the process of getting our current Offensive Agent to learn because it allowed us to avoid having our agent begin learning by performing random actions. The Offensive Agent could instead learn from our Agent0 agent that already worked well.

The structure of the Neural Network is broken down into the following components and processes:

1. Data Preparation
    a. **Module**: loadData
    b. Data is read from two pickle files: trainingInput.pickle and trainingOutput.pickle
    c. The data is dumped into ndarrays that are returned as tensors
2. Model Definition
    a. **Module**: __init__, setModel
    b. In the constructor of the class, we define a sequential model using 2 linear and a rectified linear unit function .
    c. At the moment it takes 4 inputs, 2 hidden layers, and produces 1 output
3. Model Training
    a. **Module**: Train
    b. The model needs a learning rate. We use 0.001 as the learning rate
    c. The model needs to train an X amount of times to adjust its weights and reduce loss
    d. The model uses the input(x) and output(y) to feed data into the model
    e. 10,000 gave use a plateau in the loss, which is what it is desired from training

    f. Inside the loop we forward pass the data, calculate the loss, and do a zero gradient and running a backward pass
    g. After is all been trained, gradient descent is done in order to update the weights
4. Model Prediction
    a. **Module**: predict
    b. The prediction is made by passing the four features into the model
    c. The model takes the features, runs calculation, and returns a tensor with a result based on trained data.
    d. The tensor is returned back to the agent as a 1x1 numpy array
        i. The agent uses this value to decide on the best action to take

Decision Making

The Offensive agent uses the following decision making logic to make a move on the board

1. To make a decision, the agent creates a DQNetwork object with the desired inputs, outputs, and layers to create the model
2. Upon creating the object, the data is loaded into tensors
3. The data is passed into the train model module
4. The data is trained and the agents are launched
5. The agent gathers the features at the current game state
6. The features are used in the function chooseAction where the Offensive Agent makes a decision of where to move
7. In order to make the best decision, the Offensive Agent uses the function bestActionNN
    a. The function gets the possible actions and runs it the model's sequential evaluation
        i. The features for that action are passed into the model by using the function getNetworkPrediction, who calls the DQNetwork function predict
    b. The function predicts the value based on the features passed into the model and returns the result
    c. In the end, the action with the highest value gets chosen
8. Once 20% of pellets have been eaten, the offensive agent returns to the team's side increase the team's score

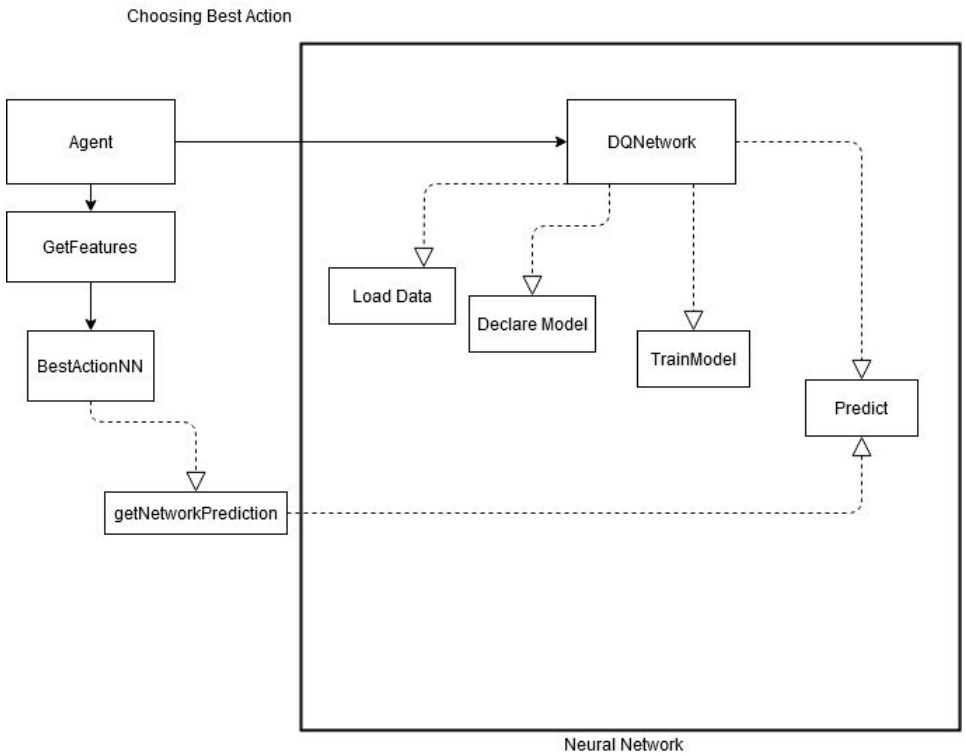


Diagram showing how the Offensive Agent decides on the action to take at the current game state and interactions with the Neural Network.

Defensive Agent

The Defensive Agent uses policies to determine the best action to take. The Defensive Agent evaluates each gameState using a linear combination of features and feature weights. These features include: number of enemies on our side, whether this agent is in the ghost state, the distance to the nearest enemy, whether this agent is at the starting position, and whether this agent is alive.

These features help the Defensive Agent figure out how to get to the nearest enemy, and helps the Defensive Agent stay in the Ghost state.

Decision Making

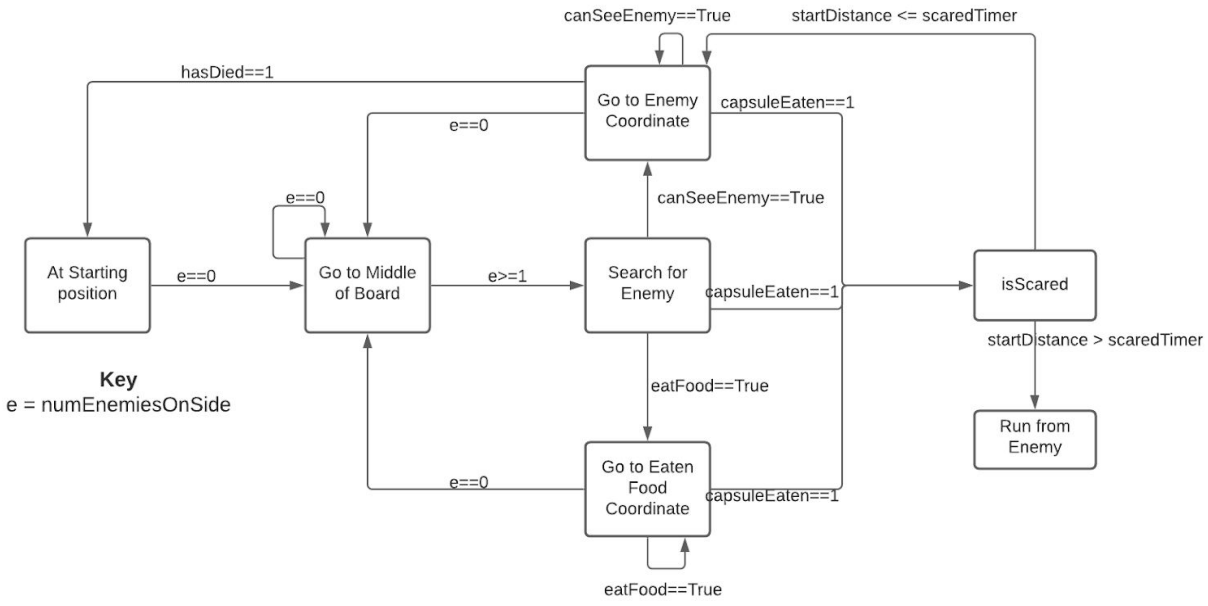
The Defensive Agent will move toward the middle of the map and wait until an enemy has entered our side. If the Defensive Agent cannot see the enemy, then it will move randomly until it sees the enemy or until the enemy eats one of the team's food pellets.

If the Defensive Agent sees an enemy, it will start chasing the enemy until the enemy is eaten. If there are no more enemies, it will return back to the middle position of the board.

If the enemy eats a food pellet, then the Defensive Agent will move to the coordinates of our team's most recently eaten food pellet. This is because it is likely that the enemy will be around the area of the most recently eaten food pellet. Food pellets are usually placed near each other, which means that the enemy is also likely to stay at that location to eat the rest of the food pellets in that location. Thus going to the most recently eaten food pellet will help the Defensive Agent locate the enemy.

Additionally, we store the most recent position of the enemy PacMan agent if it is available. This helps our Defensive Agent locate the enemy after the Defensive Agent dies. We did this because it is likely that the enemy is still near the position it was last at.

If the Defensive agent is scared, it will try to get eaten by the enemy PacMan if the distance from the start position to current position is greater than the number of remaining scared moves. Otherwise, the Defensive agent will avoid getting eaten by the enemy PacMan.



State action diagram showing decisions and actions of Defensive Agent

Evaluation

The Agents were both tested on four maps: defaultCapture, jumboCapture, fastCapture, and bloxCapture. This is because the four maps are very different in terms of size and number of pellets. We thought that if our Agents are able to perform well in these four maps, then it means that our agents are able to generalize its environment and make moves that are still optimal.

- defaultCapture: default map, agents should run well on the default map
- jumboCapture: large and a lot of pellets
- fastCapture: small and not many pellets
- bloxCapture: medium in size and number of pellets

Offensive Agent

The Offensive Agent was tested across 80 games against the Baseline team: 10 games each as a red and blue team on the four maps listed above. We also used the same number of data points (30k data points) for every game. This would ensure that the Offensive Agent has the same training data and makes similar decisions.

On the four maps, there was at least an 80% win rate. This means that the Offensive Agent does well against the baseline team when generalizing and acting in its environment.

The table below shows the team's average score across the 10 games on each map and win rate.

DQAgent VS Baseline, 10 games each map

Map	DQAgent Team	Average Score	Win Percent
defaultCapture	red	1.8	90%
defaultCapture	blue	-1.8	90%
bloxCapture	red	8.6	90%
bloxCapture	blue	-18	90%
jumboCapture	red	17.4	90%
jumboCapture	blue	-15.6	100%
fastCapture	red	0.9	80%
fastCapture	blue	-1	80%

Table of the win rate of the DQAgent team against the baseline team on defaultCapture, bloxCapture, jumboCapture, and fastCapture.

Defensive Agent

We tested the Defensive Agent's ability to defend the food by setting the Offensive Agent (or second agent in the team) to not move and stay at the starting position. This is done so that we can see if the Defensive Agent is able to defend the food well enough to get ties at the end of the match.

This was tested across 80 games against the baseline team: 10 games each as red and blue team on the four maps listed above. On the four maps, there was a 100% tie rate. This means that the Defensive Agent does well against the baseline team in preventing the opponents from obtaining food pellets and returning it back to their side. If the Defensive Agent did not defend the food pellets well, we would see that the team loses in the matches.

DQAgent VS Baseline, 10 games each map

Map	DQAgent Team	Average Score	Tie Percent
defaultCapture	red	0	100%
defaultCapture	blue	0	100%
bloxCapture	red	0	100%
bloxCapture	blue	0	100%
jumboCapture	red	0	100%
jumboCapture	blue	0	100%
fastCapture	red	0	100%
fastCapture	blue	0	100%

Table of the tie/loss rate of the Defensive Agent against the baseline team on defaultCapture, bloxCapture, jumboCapture, and fastCapture while the Offensive Agent stays at the starting position.

References

List of references used across Milestone I, Milestone II, and Final implementation.

Reinforcement Learning:

1. Hubbs, Christian. “Learning Reinforcement Learning: REINFORCE with PyTorch!” Medium, Towards Data Science, 25 Dec. 2019, towardsdatascience.com/learning-reinforcement-learning-reinforce-with-pytorch-5e8ad7fc7da0.

Neural Networks using Pytorch:

1. “Learning PyTorch with Examples¶.” Learning PyTorch with Examples - PyTorch Tutorials 1.6.0 Documentation, pytorch.org/tutorials/beginner/pytorch_with_examples.html.
2. Brownlee, Jason. “PyTorch Tutorial: How to Develop Deep Learning Models with Python.” Machine Learning Mastery, 26 Aug. 2020, machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/./
3. “Saving and Loading Models¶.” Saving and Loading Models - PyTorch Tutorials 1.6.0 Documentation, pytorch.org/tutorials/beginner/saving_loading_models.html.
4. Tabor, Phil. *Youtube*, 17 May 2019, www.youtube.com/watch?v=UlJzzLYgYoE.
5. Tabor, Phil. *Youtube*, 22 Mar. 2020, www.youtube.com/watch?v=wc-FxNENg9U.

Linear Q value:

1. cgocgo 5, and steffensteffen 9. “How to Fit Weights into Q-Values with Linear Function Approximation.” *Cross Validated*, 1 May 1965, stats.stackexchange.com/questions/187110/how-to-fit-weights-into-q-values-with-linear-function-approximation.
2. “Artificial Intelligence.” *Artificial Intelligence - Foundations of Computational Agents -- 11.3.9.1 SARSA with Linear Function Approximation*, artint.info/html/ArtInt_272.html.
3. *Going Deeper Into Reinforcement Learning: Understanding Q-Learning and Linear Function Approximation*, danieltakeshi.github.io/2016/10/31/going-deeper-into-reinforcement-learning-understanding-q-learning-and-linear-function-approximation/.