Balalayka - Final Agent

Dmitri Koltsov and Chris Daniels

Description

Our final agents named Comrades (това́рищ) are a pair of agents competing against other agents in a game of Pacman. To beat the enemy team an artificial intelligence method was used to train and test Comrades. Comrades consistently win games as the red team.

In order to beat opposing agents, several features from the `gameState` object are used. Comrades are made up of two different agents; an offensive Comrade and a defensive Comrade. The offensive Comrade eats food to score, while avoiding enemies on the hostile side of the layout. The defensive Comrade makes sure opponents do not eat our food to score.

To make Comrades consider these possibilities, a reinforced learning neural network is used to train and test. Features from the `gameState` object are the features used to calculate the best legal moves at any state. Reinforced learning is highly used for training a model while data streams. Since this pacman has options for running multiple games and training a model, reinforced learning was chosen.

Requirements

Since we are using reinforced learning to calculate the next best move for Comrades, various actions are considered positive or negative. The PyTorch package is used to generate the predictions. PyTorch has loss functions, neural networks and optimizers available to use.

The mean squared error function is being used for the loss and Adam for the optimization. Both of these are supplied by the PyTorch package. For the predictions, a Dueling Deep Q Network is used.

In order to get the agent to learn which state and actions result in the highest win rate, each Comrade must get rewarded.

Each Comrade uses the same reward structure. The reward structure has different modifiers for each Comrade. The offensive Comrade has larger modifiers that tune rewards for seeking, eating and scoring. The defensive Comrade has larger modifiers tuned for defending food on our side and eating the enemies.

Modules

The Comrades go through stages of learning what actions work well and what actions do not work well.

Stage 1: Comrades take legal actions that accrue positive rewards and avoid receiving penalties. The Comrades play the game themselves, choosing actions using the Boltzmann exploration algorithm. The Pacman game has a command line argument that runs the game `N` times. The command is `$ python capture.py --red=myTeam -n N -q` and can be used with any agent.

During this stage, the game can be run for several hundred games for each training session. Since the `-q` flag quiets output in the command to run the game, the runs are faster. Comrades learn fairly well for the first 100 games.

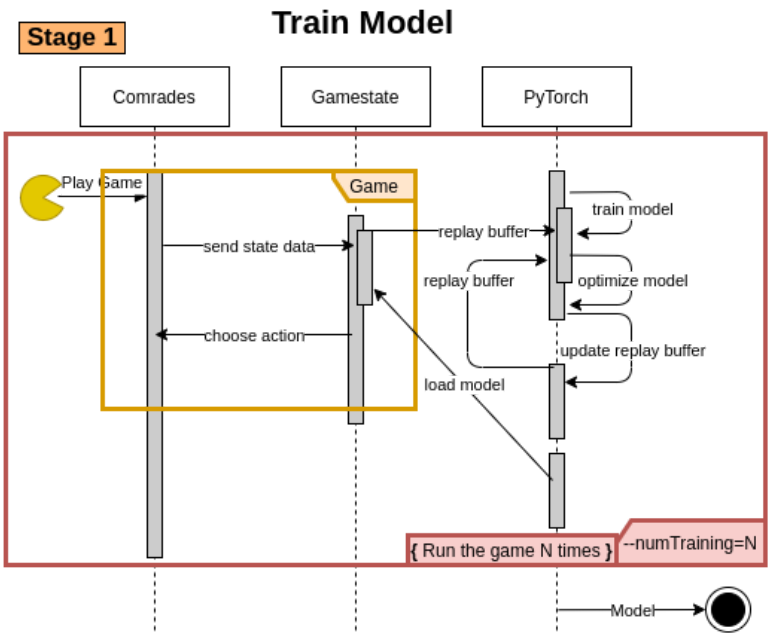
During one game data is collected on states and the corresponding action taken from this state and a reward/penalty is applied to corresponding state/action pair. After the game, all of the collected data is joined with the replay buffer data. A reinforcement learning algorithm is used to train the model using the joined data.

Below is a block of code of the reinforcement learning algorithm:

```
for epoch in range(self.epochs):
   if epoch % self.learning_step == 0:
      target_Q_network.load_state_dict(self.online_Q_network.state_dict())
      with torch.no_grad():
         online_Q_next = self.online_Q_network(next_states)
         target_Q_next = target_Q_network(next_states)
         online_max_action = torch.argmax(online_Q_next, dim=1, keepdim=True)
         y = rewards + (1 - done) * self.gamma * target_Q_next.gather(1,
online_max_action.long())
    loss = F.mse_loss(self.online_Q_network(states).gather(1, actions.long()), y)
```

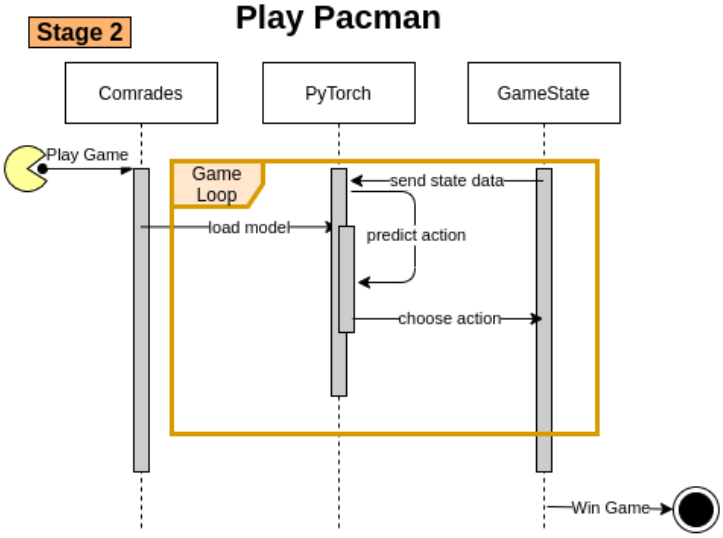
After training, the model is saved so the next game will use the updated model. We put new data from the last game into the current history. We keep up to 10,000 records in the replay buffer.

Below is the sequence diagram for how the model training flows. The game is played by Comrades, which choose an action. Each state/action pair has penalties or rewards associated with them. A replay buffer is used to sample from for each prediction. After this stage, the testing of the agent begins.



A Dueling Deep Q Network is used and has an output layer with 5 nodes for each action. The input layer takes in each `gameState` feature. The hidden layers use the hyperbolic tangent function for the activation function. For the outer layer, the linear function is used.

Stage 2: The game is played with the resulting model from stage 1. When the game is active, we observe how the Comrades play Pacman. These observations can be recorded for later consideration. The observations on how Comrades choose actions are used for stage 3.



Depending on how well Comrades play these games, determines how the rewards and penalties are shaped. Stage 2 is meant for testing the model. Stage 2 is a checkpoint on how Comrades is doing.

Stage 3: If Comrades are not doing as well in various states, revisions are taken on the reward structure. This stage is used to review how rewards are being seeked, and how penalties affect the decisions. Comrades may try to minimize the penalties, or

maximize the rewards. We want Comrades to have a healthy balance of avoiding penalties and gaining rewards.

The reward structure is adjusted according to the observations from stage 2. If adjustments are made, then Comrades return to stage 1 for training. The adjustments should allow Comrades to make decisions that win games a majority of the time.

Detailed Design

Features

To feed in data for the input layer of the Dueling Deep Q Neural Network, the various features of the `gameState` are used. Each Comrade uses this set of features. Below is a list of features grabbed for inputting into the network.

For the first four feature groups, there are a combined 25 features. Each feature has a list of either a -1, 0, or 1. For example, a future food list containing `[0, 1, 0, -1, 0]` means that index 1 has value 1 and corresponds to the 'North' action. So 'North' can take the agent closer to the food. If an index has value -1 then the agent gets further from the food. If an index has value 0, then that action is illegal from the current position. Below are the feature groups:

- Future Food positions. A list of 5 elements that correspond to the actions
- Future capsule positions. A list of 5 elements that correspond to the actions
- Future food drop positions. A list of 5 elements that correspond to the actions
- Future enemy positions. Two lists with 5 elements each that correspond to the actions.

Then there are 12 other features used. There are a total of 37 features. Below is the list of the other 12 features used:

- Agents scared timer
- Horizontal position of the agent
- Scared timer for both enemy agents
- The distance to both enemy agents
- Current score
- Distance to the power capsule
- Amount of food inside the agent
- Amount of food still on the friendly side
- Amount of food still on the enemy side
- Number for the action turn the agent is on

Rewards

Each Comrade is rewarded for:

- Eating food
- Scoring points
- Winning (removed for last version)
- Killing an opponent
- Moving toward the food
- Moving toward the drop point (with caring food presence)

Each Comrade is also penalized for:

- Dying by an opponent
- Moving away from targeted food (food on hostile side for offensive Comrade and food on home side for defensive Comrade)
- Moving away from drop point (with caring food presence)

Comrades are also rewarded for moving toward the enemy when:

- On home side and Comrade is not scared
- On hostile side and enemy is scared

Comrades are penalized for moving away from the enemy when:

- On home side and Comrade is not scared
- On hostile side and enemy is scared

Comrades are rewarded for moving away from the enemy when:

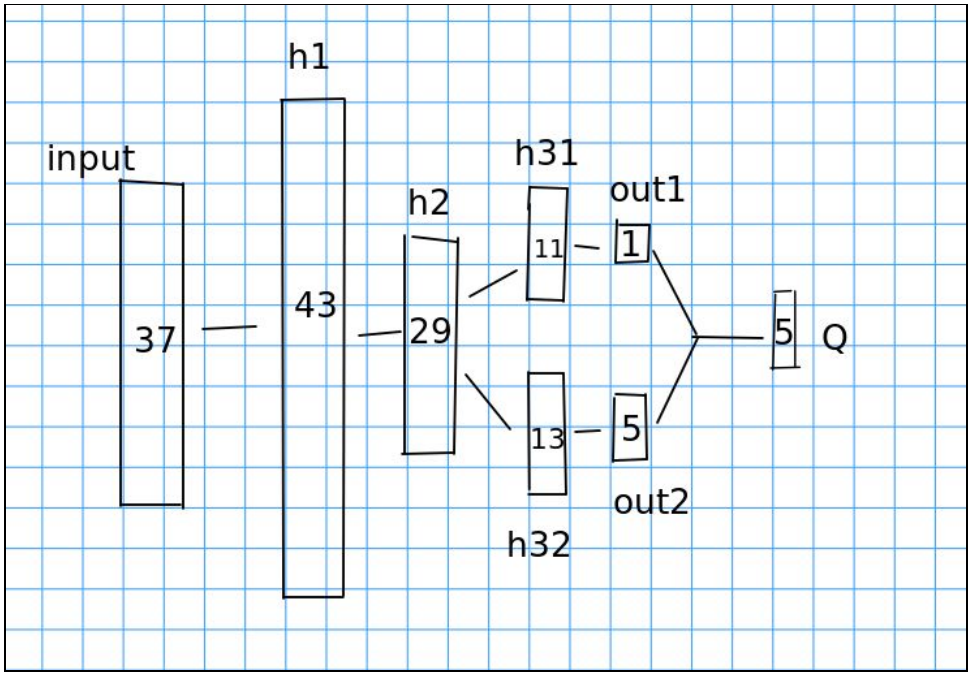
- On home side and Comrade is scared
- On hostile side and enemy is not scared

Comrades are penalized for moving toward the enemy when:

- On home side and Comrade is scared
- On hostile side and enemy is not scared

Neural Network

To get action predictions, a Dueling Deep Q Network is used. Below is an image of the network. The input layer takes in the feature set of 37 features. It has 2 hidden layers that break off into 2 output networks that are combined at the end.



As a result of using a dueling deep Q network, there are 2 intermediate output layers that get combined into one final output layer. Below is the block of code of the forward function of the dueling deep Q network.

```
def forward(self, state):
        y = self.a_func(self.fc1(state))
        y = self.a_func(self.fc2(y))

        value = self.a_func(self.fc_value(y))
        adv = self.a_func(self.fc_adv(y))

        value = self.value(value)
        adv = self.adv(adv)

        adv_average = torch.mean(adv, dim=1, keepdim=True)
        Q = value + adv - adv_average

        return Q
```

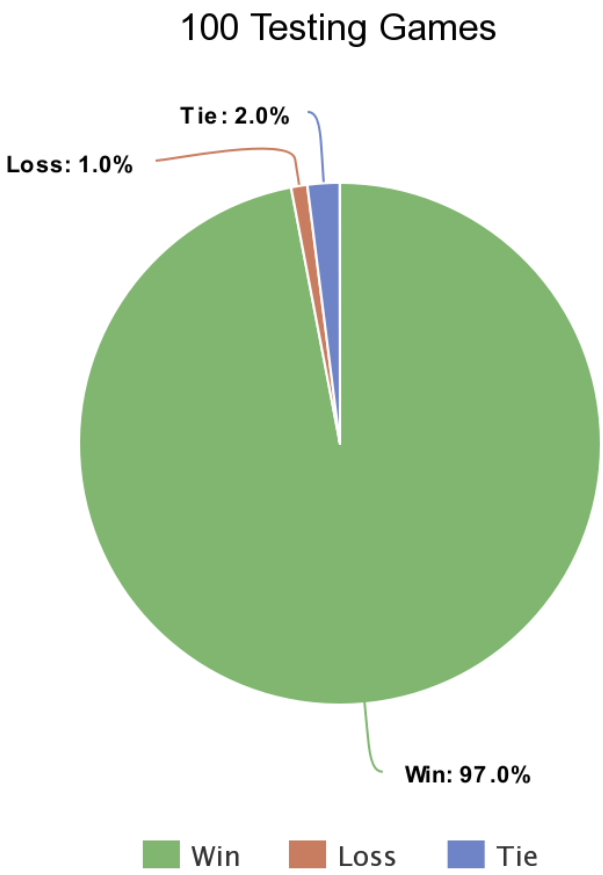
Evaluations

Model

Thousands of games were played using various rewards, penalties and features. The model statistics shown below are from playing against the baseline team, on the default maze layout. Comrades can be played on the blue team, on different layouts, but the results may vary from these.

The following pie chart is for the first 100 training games:

Further training, which was more than 1000 games increased the win rate. The win rate stayed around 80%-90%. Further tuning the model (changing gamma and reward modifiers) didn't give any stable improvements. The best model we have is the one after 870 training games, and produces these results:



Average Score: 5.2

Scores: 3, 1, 8, 7, 3, 3, 6, 9, 3, 3, 7, 7, 4, 7, 3, 5, 7, 2, 3, 4, 7, 8, 7, -11, 8, 12, 8, 1, 8, 4, 5, 3, 3, 8, 3, 4, 8, 4, 5, 3, 7, 5, 3, 3, 7, 8, 6, 3, 3, 3, 8, 8, 12, 11, 9, 12, 8, 8, 8, 12, 3, 3, 3, 4, 4, 8, 3, 3, 7, 7, 3, 2, 4, 4, 3, 8, 3, 3, 4, 3, 0, 3, 3, 8, 4, 3, 4, 7, 3, 12, 3, 2, 4, 7, 5, 6, 9, 6, 12, 0

References

- https://github.com/pytorch/pytorch
- https://pytorch.org/
- Reward Shaping - https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0
- PyTorch Tutorial - https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
- Diagrams - https://draw.io
- Charts - https://www.meta-chart.com/