State based Strategy Rough Draft

Deep Learning residual network model ResNet

Use ResNet(Residual Networks, deep convolutional neural network) and implement DAGGER (Data Aggregation from Diverse Demonstrations to Guide Exploration and Reinforcement Learning), and reinforcement learning as base agent to learn control policies. ResNet uses residual blocks (skip connections).

Overview:

- Imitation learning with additional DAGGER training. First collect training data of the Jurgen agent playing against itself and other opponents. Plat against multiple opponents to get a wide variety of state/action pairs. Delete the games in which the Jurgen agent does not win.
- Use imitation learning to create a model that copies Jurgen. When the model is trained, use DAGGER to improve gameplay. Play more games with our state agent against opponents. Training loop revisits these games and compares Jurgens actions to our models actions to generate loss values.
- Input will be the state information of the game (positions of player's kart and ball). Ignore opponent's states for input. Output will be action is value for braking, steering and accelerating.

Step-by-Step Guide:

- 1. Collect Training Data:
- Initially, collect a dataset of state-action pairs by observing expert gameplay or using a simple heuristic agent.
- Run the Jurgen agent in the game environment and record the states observed and the actions taken by the agent in response to those states.

2. Train Initial Model:

- Train a neural network model using the collected dataset of state-action pairs. Switch sides after each epoch for more stable training results.
- The model takes the game state as input and outputs the corresponding action to be taken by the agent.

3. Iterative Training with DAGGER:

- Use the trained model to play the game in the environment.
- At each step, observe the state of the game and let the model predict an action.
- Instead of using the model's action directly, incorporate Jurgen expert feedback to determine the correct action.
- Achieved by comparing the model's predicted action with the expert's action
- Collect this new data (state agent and expert Jurgen actions) and aggregate it with the existing dataset.
- Retrain the model using the combined dataset.
- Repeat this process iteratively, refining the model with each iteration.

4. Fine-Tuning with Reinforcement Learning:

- After several iterations of DAGGER, we can fine-tune the model using reinforcement learning techniques.
- Set up a reinforcement learning environment where the state agent interacts directly with the game environment.
- Initialize the agent's policy with the model obtained from DAGGER and continue training using reinforcement learning to improve performance.

5. Evaluation and Deployment:

- Evaluate the trained agent in the game environment to assess its performance.
- Monitor metrics such as the number of goals scored, matches won, and overall gameplay quality.
- Iterate on the training process, adjusting hyperparameters, model architecture, or training strategies as needed to improve performance.

Setup

- Input to model is features of player state and puck state as did Jurgen, do not expand feature dimension. (features tensors include: kart centers, kart angle, kart to puck angle, center of both goal lines, difference between kart angle and kart to puck angle, puck location, difference between puck center and goal line)
- Make sure to fix data shuffling and training by collection action state pairs from both sides of the field (red and blue).
- Focus on copy Jurgen agent behavior, DAGGER training on good games (games wons).

- For **acceleration** use SmoothL1 loss function as it is treated as regression. SmoothL1 is designed to be less sensitive to outliers compared to the mean squared error (MSE) loss function, particularly when dealing with data that may contain noisy or inconsistent targets.
    - Any acceleration values outside of the range [0,1] are clipped to 0 or 1.

- **Steering and braking** are treated as a classification problem. Use Binary Cross Entropy (BCE) with logits loss. Propagate the outputs of the model through a sigmoid function to distribute the values from 0 to 1
    - Steering: the two possible classifications are 0, mapped to a steering of -1, and 1, mapped to a steering of 1. The sum of the individual losses is then back-propagated through the network.
    - Braking: model brakes if the activation value is below 0.5, and does not if the value is above.

Data Generation and Training

- Data generated in training loop by letting state agent play against provided TA agents. Use Jurgen agent for expert in DAGGER. Dagger to overcome mismatches in the training and testing distribution.
- Perhaps include dummy agent to go against which does not do anything so our agent can learn to score a goal without interference.
- First epoch, collect game data where we let Jurgen expert play the game against the dummy agent to quickly learn to drive to the puck to score a goal. Rest of epochs use DAGGER where we allow state_agent to play the game instead of Jurgen.
    - Every state that state_agent encounters, collect action that Jurgen agent would do. Append this data to the dataset to be used in future epochs.