TEXAS
The University of Texas at Austin

TOPIC 4 REINFORCEMENT LEARNING

Policy Gradients

- Policy gradients trains a classification NN to tell you what the probability of the best action is for each state
- Traditionally, to train a classification model we
  - $\min \sum_i loss(truth_i, predicted_i)$
- Loss may be categorical cross entropy
- We don't know the truly best action!
- This is the same problem as DQN!

Policy Gradients

- To address this, let's simulate a bunch of actions randomly and replace the truly best action with just the actions we actually took
  - $\min \sum_i w_i loss(actual\ action_i, predicted_i)$
- Here w_i is a weight for each action we actually took
  - Did an action that we took eventually lead to a high reward or low reward?
  - We put higher importance on remembering actions that lead to high rewards!
  - If the eventual reward was bad, we can let $w_i < 0$
  - w_i should be related to discounted reward (value function)

Policy Gradients

- For us, w_i will be the discounted eventual reward for each point
- Suppose we score a point in frame 200
  - $w_{200} = 1, w_{199} = \delta, w_{198} = \delta^2, \ldots$
- Suppose we lose a point in frame 100
  - $w_{100} = -1, w_{99} = -\delta, w_{98} = -\delta^2, \ldots$
- We can do this because we wait until after a game is over to assign the weights

Policy Gradients

- Randomly initialize a neural net for classification
  - Input state, output probability of each possible action
- Simulate the environment forward in time
  - Use NN to decide action at each state – push buttons according to output probability
  - Save frames/actions/reward at each time
  - Calculate the weights
- Do a little bit of SGD on weighted loss to update NN parameters a little bit

Policy Gradients

- Let's play Atari!

- Again, this code probably won't work very well
- We'll get into why later
- It will also be slow!

Policy Gradients

- I sort of lied to you when I said this is called Policy Gradients
- In truth, Policy Gradients does some complicated math the find the gradient vector of all the weights and biases that would lead to a better classification NN without using this specific weighted loss function
- The general idea is similar though…

Memory Buffers

- One of the issues with Q-learning and policy gradients, as written, is that there's lots of correlation between states
  - Two adjacent frames are mostly the same pixels!
  - Both frames were used for training an epoch
- One solution to this problem is called a memory buffer
  - Keep lots of games in memory
  - When you train, randomly grab an assortment of frames from different games in memory
  - Periodically remove some old games from memory

Memory Buffer

- Store up to 100 (arbitrary) games of frames, actions, rewards
- Play a game, add to the buffer, remove oldest game if more than 100 stored
- After the game, randomly pick 1000 (arbitrary) of the frames/actions/rewards from the 100 games in buffer
  - Grab several frames (3-4) before each one to create the history, and the frame after to get the ‘truth’
  - Generate the ‘truth’ for each random frame
  - Take 1 sgd epoch with those frames

Annealing

- In our Q-learning we can also use a linear annealed policy, like in the HW
- Start with a large ϵ and play a few games
  - Warm up
  - Maybe don't learn while you're warming up?
  - Build up your memory buffer
- Then gradually reduce ϵ after each game
  - Until you hit some lower limit of ϵ

Improve Performance

- Nothing we've done so far works very well in pong ☹
- There are a few structural issues with pong
  - It takes ~20 frames at the beginning for the ball to appear
  - Similar after each point is scored
  - Try removing some of those frames from learning
- The DeepMind people don't pick an action every frame
  - After 4 frames pick an action
  - Play the same action for 4 frames in a row after that
  - Then pick another action

Improve Performance

- In PG we used the true discounted reward to evaluate our performance
- Could we do this in a variant of Q-learning
  - When generating the truth don't use $r_t + \max_x \delta v(S_{t+1}, t+1)$
  - Instead use the actual discounted reward at the end of the point, as in PG
- Who knows if this will be any better…give it a shot

Actor-Critic Methods

- One new strategy is to combine DQN with PG
- In PG we used the true discounted reward as our weight for the loss
- The actor-critic method uses the estimated value function from DQN as the weight for the loss function
  - Use PG to pick actions that get chosen
  - Use DQN to evaluate if the actions are good or not
    - Weight in the objective
- Train both networks simultaneously

Actor-Critic Methods

- This is advantageous because it helps both networks decouple acting and learning
- Both acting and learning can be more focused!
- Q-learning sometimes has a bias issue when you are training and using the NN to pick your action
- Policy gradients should look at the expected future reward, instead of the actual future reward of the particular sample path!

Philosophy of RL

- There are people in the computer science community that are interested in general purpose RL (intelligence?)
  - Using the same NN to solve many different problems
  - Many books teach this principle
  - DeepMind is really interested in this!
- While I think this is theoretically interesting, as a business professor I'm way more interested in tailoring my RL to fit specific applications
  - Why should the same NN that does well at pong also do well at scheduling delivery trucks?
  - If I'm going to use RL to complete a task that helps my company make money, I'm may only be interested in that specific task, I may not need it to do well on other tasks too!

Dueling Networks

- A recent advance in RL is to train 2 NNs and have them play against each other
  - Dueling networks
- When NN1 makes a decision, it knows the distribution of NN2's actions at this state
- NN1 optimizes according to what it knows NN2 will do
- The same is true for NN2
- We'll see more of this when we go back to DP