TEXAS
The University of Texas at Austin

TOPIC 4 REINFORCEMENT LEARNING

Reinforcement Learning

1. Initialize the NN
2. Play pong for a while and do what the NN tells us (EvE)
3. Remember everything
4. Periodically use our memory to take a few sgd steps to make the NN a little bit better
   - This is the tricky part!
5. Hopefully after long enough the NN is pretty good?

Reinforcement Learning

- There are several popular methodologies for RL
  - Q-Learning
  - Policy Gradients
- These two methods approach the problem very differently!

Q-Learning

- Approximate the value function using a neural network
  - Input of neural network is the state
  - Output is value function that corresponds to each possible action
    - If there are k possible actions, then output layer of NN has k nodes
  - Choose the action with highest value function
    - Exploration vs exploitation! – usually ϵ-greedy

Policy Gradients

- In policy gradients we change the way we use the value function
- We don’t explicitly try to optimize the value function
- Instead, we pose the problem of picking the optimal action as a classification problem
- The classification model takes the state/frames as the input and outputs the probability of each action being the best

Reinforcement Learning

- In both cases the general algorithm is similar

1. Initialize NN with random weights
2. Define a loss function that will (hopefully) lead to more good actions
3. Interact with the environment via simulation (play pong)
4. Evaluate the NN at the state to pick an action while considering exploration vs exploitation
5. Do a little optimization (sgd) to get ‘better’ weights
6. Repeat 3-5 over and over

Deep Q-Learning

- Recall the Bellman Equation
  - $v(S_t, t) = \max_x E[r_t + \delta v(S_{t+1}, t+1)]$
- Remember, we want to get the approximate value of v, for each action, from the NN
- Q-Learning picks NN's weights and biases to minimize
  - $\sum_{s \text{ in memory}} \left(v(S_t, t) - r_t - \max_x \delta v(S_{t+1}, t+1)\right)^2$
  - This is called Temporal Difference (TD)
    - Value function today minus value function tomorrow
- But it's a bit complicated…

Deep Q-Learning

- Remember, DQN wants to learn what the value function would be for each possible action
- Why isn't r_t inside the max?
  - We train using the action we actually played
  - The max for next period is what we think will be best..
- Each time we interact, we'll look at potential VF from each action and pick the best
- Early in learning, we'll often pick the wrong action!
- Eventually after enough learning, when we pick the 'best' hopefully it really will be best

Deep Q-Learning

- Remember the mining problem
- Even if we didn't extract 2 tons of ore at a particular time, we still needed to know what the value function would have been if he had!
- By picking the wrong action sometimes, we increase our ability to know what the value function would be

Deep Q-Learning

- $\left(v(S_t, t) - r_t - \max_x \delta v(S_{t+1}, t+1)\right)^2$

- SARSA – State Action Reward State Action
- To even evaluate that objective at a particular state you need
  - Today's state (for the first v)
  - Today's action (for the first v)
  - Today's reward (for the r)
  - Tomorrow's state (for tomorrow's v)
  - Tomorrow's action (don't really need this for Q-learning…)

Deep Q-Learning

- We want the NN to give us v for each possible action
- NN's work like regression
  - $\min \sum (predicted \ - truth)^2$
- $v(S_t, t)$ is like the predicted value given a state
- $r_t + \max_x \delta v(S_{t+1}, t+1)$ is like the truth
- Just like regression, we just need to give TF the vector of truths and the corresponding states, and it handles the rest…but…

Deep Q-Learning

- We give TF the vector of states/frames, S_t, and the vector of 'truths'
- We don't know the truths
  - To get the truths we must have r_t and $v(S_{t+1})$
- We get r directly from Atari; after I click a button, it tells me what the reward is for clicking that button
- To get $v(S_{t+1})$ we evaluate the NN at the S_{t+1} frame
  - This gives us the value for each possible action at t+1
  - Just take the biggest one!

Deep Q-Learning

- There's 1 more issue…The NN output layer has as many nodes as there are possible actions
- TF treats the objective as
  - $\sum_{data} \sum_{output\ nodes} (predicted\ - truth)^2$
- When we play pong, we actually only took 1 of those possible actions
- We don't know what the next state would have been if we had taken a different action
- We only know the truth for one of the output nodes

Deep Q-Learning

- One solution to this is to add weights to the objective function
  - $\sum_{data} \sum_{output\ nodes} w_{d,o}(predicted\ - truth)^2$
- For each data point
  - w is 1 for the output node that corresponds to the action we actually took
  - w is 0 for all other output nodes
- Then it doesn't matter what the 'truth' is for the other output nodes
- Doing this in TF is a little tricky

Deep Q-Learning

- Let's give it a try…
- https://arxiv.org/abs/1312.5602

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

Deep Q-Learning

- Technically, the code we just saw was a *Double* Deep Q-Network
- To be just a simple Deep Q-Network we would take an SGD step after each frame was played

- Double Deep Q-Networks use one network to estimate the truth, while learning on another network
  - Periodically update the truth giving network
  - This is exactly what we did: find the truth for every frame using the old network weights then run several SGD steps to update the weights