50.021 – AI

Alex

Week 09: Reinforce with baseline

1 Reinforce with a baseline

You can start with your deep-Q-code for the cartpole.

Implement Reinforce with a baseline. That is simpler actually than DQN. Run it for cartpole and lander.

- **Network:** suggestion for your network: it needs to have one fc-layer with softmax for action probability π, and one fc-layer with only one real-valued output for the v_π estimate. Both of these fcs are set over the same feature map. For the shared features below you can use one layer with 128 (often good for the lander, more stable) or two layers with 24 and 24 . You can use leaky relu.
- **actions:** `select_action(...)` needs to be modified so that now you do: $a_t \sim \pi(a|s_t)$ with ϵ probability for a random action. Not a as argmax with ϵ probability for a random action. torch.categorical can help. You can start with a random ϵ of only 0.25 or 0.05 (same as end-ϵ), since reinforce is on-policy.
- **learning I** throw away the replay memory, empty `optimize_model(...)` of its contents as a whole. nothing of this will be kept. Your new `optimize_model(..)` will take as inputs for learning a list of states, a list of rewards, and a third list of what ?
- **learning II**: play one episode using $a_t \sim \pi(a|s_t)$ with `select_action(...)` until it terminates. creates these three lists. pass them to `optimize_model(...)`.

in `optimize_model(...)` implement reinforce with a baseline. as spoken in class, this needs you to define a loss function which is a sum of two loss components. You can optimize all parameters at the same time. For the value estimate retraining I used huber loss aka smooth L1.

Be careful when to use `with torch.no_grad()` or `tensor.detach()` and when not.

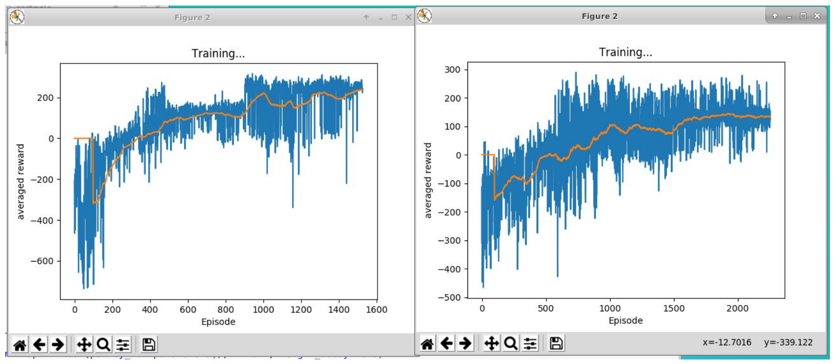
As for computing returns at step t for use with the loss it is practical to iterate over the lists backwards in time:

$$G_t = R_t + \gamma G_{t+1}$$

After you got your loss, do the usual single-parameter-set-optimization triad: `optimizer.zero_grad(), loss.backward(), optimizer.step()`

Note the difference to DQN: in reinforce you optimize once per episode. in DQN once every step of an episode.

- some parameter advice: I use $8e-3$ for learning rate in RMSProp. I dont claim my settings are optimal.



Lander left with 1 hidden layer of 128, right with 2 hidden layers of 24 each.

- run reinforce with baseline on cartpole, report graphs with average rewards over last 100 episodes
- run reinforce with baseline on lander, report graphs with average rewards over last 100 episodes
- run reinforce without baseline on lander, report graphs with average rewards over last 100 episodes. For that you need to create a network without value output.

Optional, gives you not more points: you can try replay memory with likelihood weighting, or n-step A2C (which will need a Q-estimate!!)

Short episodes of catastrophic unlearning can happen X-D.

If you have the patience, once every 50 or 100 episodes of your lander run for every step in your episode an `env.render()` to see what the lander is doing.