Deep Reinforcement Learning in R with rlR package [1]

Xudong Sun

July 4, 2018

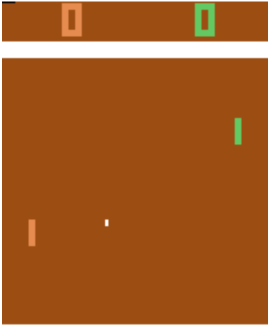
[1] https://github.com/smilesun/rlR

Teach computer to play games with computer


```r
library(rlR)
env = makeGymEnv("Pong-v0")
env$overview()

##
## action cnt: 6
## state dim: 210, 160, 3
## discrete action
```

Teach computer to play games with computer

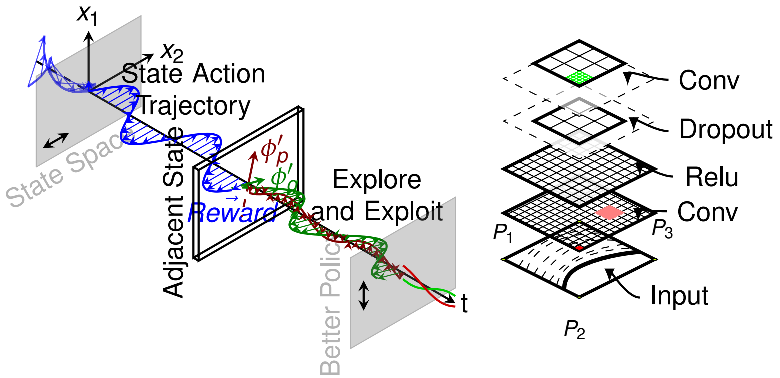

```
env$snapshot(steps = 25)
```



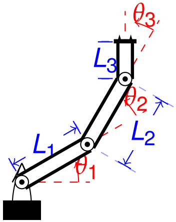
Available Environments in rlR

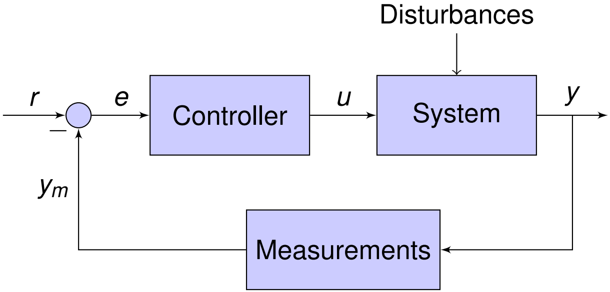

```r
library(rlR)
listGymEnvs()[1:30]

##  [1] "CartPole-v0"                  "CartPole-v1"
##  [3] "MountainCar-v0"               "MountainCarContinuous-v0"
##  [5] "Pendulum-v0"                  "Acrobot-v1"
##  [7] "LunarLander-v2"               "LunarLanderContinuous-v2"
##  [9] "BipedalWalker-v2"             "BipedalWalkerHardcore-v2"
## [11] "CarRacing-v0"                 "Blackjack-v0"
## [13] "KellyCoinflip-v0"             "KellyCoinflipGeneralized-
## [15] "FrozenLake-v0"                "FrozenLake8x8-v0"
## [17] "CliffWalking-v0"              "NChain-v0"
## [19] "Roulette-v0"                  "Taxi-v2"
## [21] "GuessingGame-v0"              "HotterColder-v0"
## [23] "Reacher-v2"                   "Pusher-v2"
## [25] "Thrower-v2"                   "Striker-v2"
## [27] "InvertedPendulum-v2"          "InvertedDoublePendulum-v2
## [29] "HalfCheetah-v2"               "Hopper-v2"
```

RL as a Functional Optimization process



Classical Control Problem



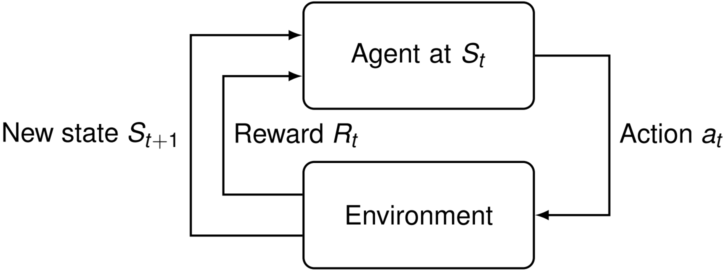


CliffWalker[2]

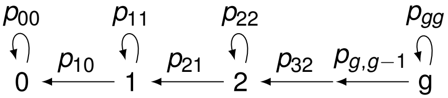

```
##
##    v v v > v v v v v v > v
##    v v v v v v v v v v > v
##    > > > > > > > > > > > v
##    ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
##
##    > > > > > > > > > > > v
##    > > > > > > > > > > > v
##    > > > > > > > > > > > v
##    ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
```

[1] Sebastian Gruber

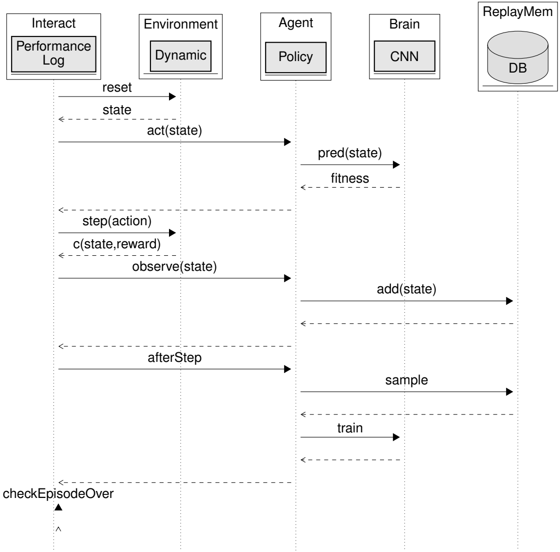
Environment Agent Interaction through Policy



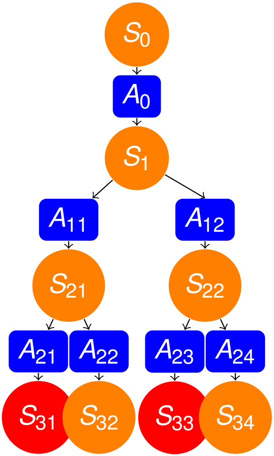
- Agent Environment Interaction
- State s, Action a, Transition(Environment *MDP*), Reward R
- Learning reaction Policy(e.g. Look Up Table) $\pi(a|s)$
- Returns: accumulated gain of reward $G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i}$

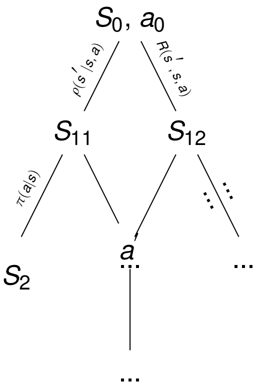


rlR sequence diagram

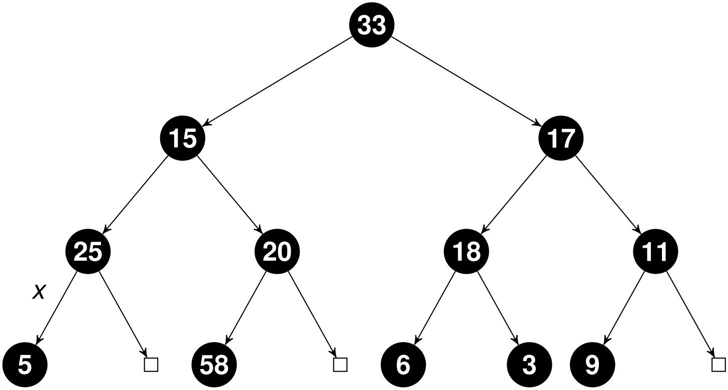


BackUp Diagram and Policy Environment Uncertainty





Long Term Consideration



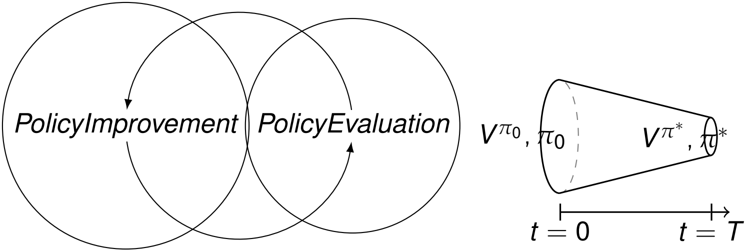
State Value Function and Bellman Equation

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}[\sum_{i=0}^{\infty} \gamma^{i} R_{t+i} | S_t = s] \\
&= E_{\pi}[R_t + \sum_{i=1}^{\infty} \gamma^{i} R_{t+i} | S_t = s] \\
&= E_{\pi}[R_t + \gamma \sum_{i=1}^{\infty} \gamma^{(i-1)} R_{t+i} | S_t = s] \\
&= E_{\pi}[R_t + \gamma \sum_{i'=0}^{\infty} \gamma^{i} R_{t+1+i'} | S_t = s] \\
&= E_{\pi}[R_t + \gamma V^{\pi}(S_{t+1}) | S_t = s]
\end{aligned}
$$

State (Action) Value Function and Bellman Equation

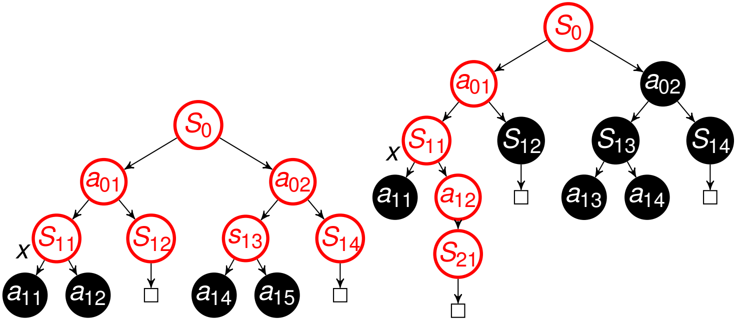
- $Q^{\pi}(s,a) = E_{\pi,\varepsilon}[R_t + \sum_{i=1}^{\infty} \gamma^i R_{t+i} | S_t = s, A_t = a]$
- $V(s_t) = \sum_a \pi(a|s_t) Q(s_t, a)$ since $\sum_a \pi(a|s_t) V(s_{t+1}) = V(s_{t+1})$
- $\pi^* = argmax_{\pi} V^{\pi}(s) = argmax_{\pi} E_{\pi}[R_t + \gamma V^{\pi}(S_{t+1}) | S_t = s], \forall s \in S$ (Otherwise replace to the better action at step t)
- $V^{\pi^*}(s) = E_{\pi^*}[R_t + \gamma V^{\pi^*}(s_{t+1}) | S_t = s]$(optimal act each step)
- $Q^{\pi^*}(s,a) = E_{\pi^*,\varepsilon}[R_t + \sum_{i=1}^{\infty} \gamma^i R_{t+i} | S_t = s, A_t = a]$
- $V^{\pi^*}(s) = max_a Q^{\pi^*}(s,a)$
- $Q^{\pi^*}(s,a) = E_{\varepsilon,\pi^*}[R_t] + \gamma \max_{a} \{Q^{\pi^*}(s_{t+1}, a)\}$

GPI



Dynamic Programming and Monte Carlo

$$DP : V(S_t) = E_\pi[R_t + \alpha V(S_{t+1})]$$
$$MC : V(S_t) = V(S_t) + \alpha(R_t - V(S_t))$$



TD(λ) algorithm and (stochastic or deterministic)Policy Gradient

$$Q^{\pi^*}(s,a) = E_{\varepsilon,\pi^*}[R_t] + \gamma \max_a \{Q^{\pi^*}(s_{t+1},a)\}$$

$$\delta = Q^w(s,a) - ([R_t] + \gamma \max_a \{Q^w(s_{t+1},a)\})$$

$$\nabla_\theta v_\pi(s) =$$

$$\sum_a [\nabla_\theta \pi_\theta(a|s) q_\pi(s,a) + \pi_\theta(a|s)[\sum_{s'} p(s^{'}|s,a) + (\gamma \nabla_\theta v_{\pi(\theta)}(s^{'}))]]$$

Implemented Algorithms in rlR

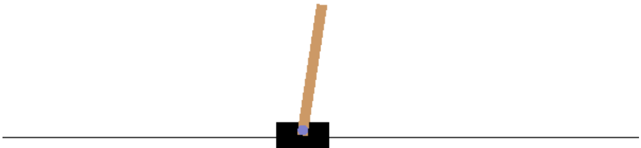
```
env = makeGymEnv("CartPole-v0")
rlR::listAvailAgent(env)

## $AgentDQN
## [1] "Deep Q learning"
##
## $AgentFDQN
## [1] "Frozen Target Deep Q Learning"
##
## $AgentDDQN
## [1] "Double Deep QLearning"
##
## $AgentPG
## [1] "Policy Gradient Monte Carlo"
##
## $AgentPGBaseline
## [1] "Policy Gradient with Baseline"
##
## $AgentActorCritic
## [1] "Actor Critic Method"
```

Experiment reproducibility in rlR

```
rlR::showDefaultConf()

##
## render                                    FALSE
## log                                       FALSE
## console                                   FALSE
## agent.gamma                                0.99
## agent.flag.reset.net                       TRUE
## agent.lr.decay                0.999000499833375
## agent.lr                                  0.001
## agent.store.model                         FALSE
## agent.clip.td                             FALSE
## policy.maxEpsilon                          0.01
## policy.minEpsilon                          0.01
## policy.decay                                  1
## policy.decay.type                    geometric1
## policy.aneal.steps                        1e+06
## policy.softmax.magnify                        1
```

rlR on Inverted Pendulum

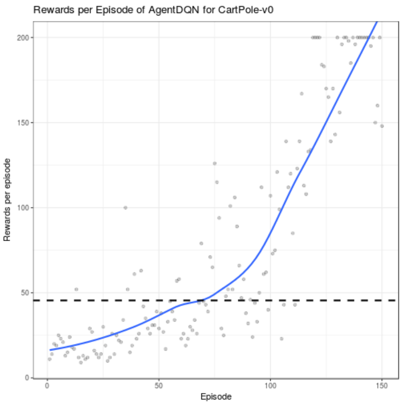

```
env = makeGymEnv("CartPole-v0")
env$snapshot()
```



rlR on Inverted Pendulum

```
library(rlR)
env = makeGymEnv("CartPole-v0")
conf = getDefaultConf("AgentDQN")
agent = makeAgent("AgentDQN", env, conf)
perf = agent$learn(200)
perf$plot()
```

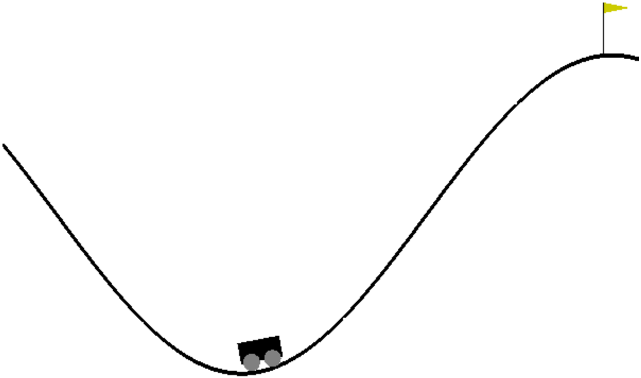
Deep Q Learning on CartPole-v0



Custom Neural Network

```
env = makeGymEnv("MountainCar-v0",
  act_cheat = c(0, 2))
conf = getDefaultConf("AgentDQN")
conf$set(console = TRUE, render = TRUE,
  policy.maxEpsilon = 0.15, policy.minEpsilon = 0,
  policy.decay = 1.0 / 1.01, replay.batchsize = 10,
  replay.epochs = 4, agent.lr.decay = 1,
  agent.gamma = 0.95)
  agent = makeAgent("AgentDQN", env, conf)
env$overview()

##
## action cnt: 2
## state dim: 2
## discrete action
```

Custom Neural Network



Custom Neural Network


```
mfun = function(state_dim, act_cnt) {
requireNamespace("keras")
    model = keras::keras_model_sequential()
      model %>%
        layer_dense(units = 10, activation = "relu",
          input_shape = c(state_dim)) %>%
        layer_dropout(rate = 0.25) %>%
        layer_dense(units = act_cnt,
          activation = "linear")
      model$compile(loss = "mse",
        optimizer = optimizer_rmsprop(lr = 0.001))
      model
  }
  agent$customizeBrain(value_fun = mfun)
  agent$learn(500L)
```

Custom Neural Network


```
conf = getDefaultConf("AgentFDQN")
conf$set(replay.batchsize = 32, replay.freq = 4L,
  console = TRUE,
  agent.lr.decay = 1, agent.lr = 0.00025,
  replay.memname = "UniformStack", render = FALSE,
  policy.decay = exp(-2.2 / 1e6),
  policy.minEpsilon = 0.1,
  agent.start.learn = 5e4, replay.mem.size = 1e6,
  log = FALSE,
  agent.update.target.freq = 10000L, agent.clip.td
  policy.decay.type = "linear")
env = makeGymEnv("KungFuMaster-v0", observ_stack_le
agent = makeAgent("AgentFDQN", env, conf)
```

Custom Neural Network

Custom Neural Network


```
pong_fun = function (state_dim, act_cnt) {
model <- keras_model_sequential()
model%>%
layer_conv_2d(filter = 32, kernel_size = c(8,8), st
  padding = "same", input_shape = state_dim) %>% la
layer_conv_2d(filter = 64, kernel_size = c(4,4), st
layer_activation("relu") %>%
layer_conv_2d(filter = 64, kernel_size = c(3,3),
  strides = c(1,1), padding = "same") %>%
layer_activation("relu") %>%
layer_flatten() %>% layer_dense(512) %>%
layer_activation("relu") %>% layer_dense(act_cnt) %
layer_activation("linear")
opt = optimizer_rmsprop(lr = 0.00025)
model %>% compile(loss = "mse", optimizer = opt, me
return(model)
}
```

Custom Neural Network

```
agent$customizeBrain(value_fun = pong_fun)
agent$learn(5000)
```

Custom Neural Network


```
rlR::listAvailConf()

##  [1] "render"                      "log"
##  [3] "console"                     "agent.gamma"
##  [5] "agent.flag.reset.net"        "agent.lr.decay"
##  [7] "agent.lr"                    "agent.store.mod
##  [9] "agent.update.target.freq"    "agent.start.lea
## [11] "agent.clip.td"               "policy.maxEpsil
## [13] "policy.minEpsilon"           "policy.decay"
## [15] "policy.decay.type"           "policy.aneal.st
## [17] "policy.softmax.magnify"      "replay.batchsiz
## [19] "replay.memname"              "replay.mem.size
## [21] "replay.epochs"               "replay.freq"
```

Thanks for your attention!

- URL: https://github.com/smilesun/rlR
- BugReports: https://github.com/smilesun/rlR/issues
- `devtools::install_github("smilesun/rlR", dependencies=TRUE)`
- Install, try it out, and have fun!