Reinforcement Learning

Prof. Christian Bauckhage

lecture 11

value- and quality-function approximation

in this lecture, we finally address the question of

> "what to do in settings where *tabular reinforcement learning* can't work ?"

to begin with, we will discuss the ideas of *value-* and *quality-function approximation*

for the fun of it, we then have a closer look at Google Deepmind's *deep Q-learning* of control policies for classic ATARI arcade games ...

outline

recap

Q-learning in stochastic environments

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right]$$

$$\Leftrightarrow \qquad \hat{Q}(s,a) \leftarrow \left[1 - \alpha \right] \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') \right]$$

Q-learning in deterministic environments

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_{sa},a')$$

Q-learning from episodes

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$

$$\hat{Q}(s,a) = \begin{cases} 0 & \text{if } s \in \mathcal{F} \\ rnd & \text{otherwise} \end{cases}$$

for many, many trials

sample $s \in \mathcal{S} \setminus \mathcal{F}$

while $s \notin \mathcal{F}$

sample $a \in \mathcal{A}(s)$ ϵ-greedily (using $\hat{Q}$)

$s' \leftarrow Succ(s,a)$

$r \leftarrow Util(s')$

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)\right]$$

$s \leftarrow s'$

Q-learning from transitions

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$

$$\hat{Q}(s,a) = \begin{cases} 0 & \text{if } s \in \mathcal{F} \\ rnd & \text{otherwise} \end{cases}$$

for many, many trials

sample $s \in \mathcal{S} \setminus \mathcal{F}$

sample $a \in \mathcal{A}(s)$ ϵ-greedily (using $\hat{Q}$)

$s' \leftarrow Succ(s,a)$

$r \leftarrow Util(s')$

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)\right]$$

SARSA learns *on-policy*

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$

$$\hat{Q}(s,a) = \begin{cases} 0 & \text{if } s \in \mathcal{F} \\ rnd & \text{otherwise} \end{cases}$$

for many, many trials

sample $s \in \mathcal{S} \setminus \mathcal{F}$

sample $a \in \mathcal{A}(s)$ ϵ-greedily ($\Leftrightarrow$ using $\hat{Q}$)

while $s \notin \mathcal{F}$

$s' \leftarrow Succ(s,a)$

$r \leftarrow Util(s')$

sample $a' \in \mathcal{A}(s')$ ϵ-greedily ($\Leftrightarrow$ using $\hat{Q}$)

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\Big[r + \gamma\,\hat{Q}(s',a') - \hat{Q}(s,a)\Big]$$

$s \leftarrow s'$

$a \leftarrow a'$

Q-learning learns *off-policy*

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$

$$\hat{Q}(s,a) = \begin{cases} 0 & \text{if } s \in \mathcal{F} \\ rnd & \text{otherwise} \end{cases}$$

for many, many trials

sample $s \in \mathcal{S} \setminus \mathcal{F}$

while $s \notin \mathcal{F}$

sample $a \in \mathcal{A}(s)$ ϵ-greedily (using $\hat{Q}$)

$s' \leftarrow Succ(s,a)$

$r \leftarrow Util(s')$

$a' \leftarrow \max_{a''} \hat{Q}(s',a'')$

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\Big[r + \gamma\,\hat{Q}(s',a') - \hat{Q}(s,a)\Big]$$

$s \leftarrow s'$

observe

everything we have been doing lately could have been done using (Monte Carlo) tree search !

question

so why all the fuzz about reinforcement learning ?

answer

consider this . . .

note

state and/or action spaces can be large

last time, we worked around these issues, however . . .

in many situations, state and/or action sets are way too large to ever be *practically* tabulated, for instance

> the game of *backgammon* has an estimate of 10^{20} states
> the game of *go* has a staggering estimate of 10^{170} states

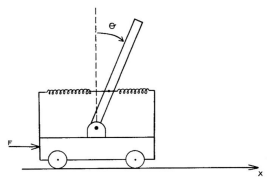
$\Leftrightarrow$ it is categorically impossible to learn policies for these games using the methods we discussed so far

note

state and/or action spaces can be continuous

consider, for instance, the *cart pole* control problem due to

D. Michie, R.A. Chambers, *BOXES: An Experiment in Adaptive Control*, Machine Intelligence 2, **1968**



states $s = [\theta, x, \dot{x}, \ddot{x}]$

$\Leftrightarrow$ (without discretization) it is impossible to address continuous state / action spaces using the methods we discussed so far

good news

RL “easily” extends to large or continuous environments or tasks

however, this requires stepping out of our comfort zone

up until now, we just thought of value- and quality functions $V_\pi(s)$ and $Q_\pi(s, a)$ as lookup tables or arrays

to address environments / tasks with too many states or actions, we shall next consider approaches not at all alien to machine learners, namely . . .

approximating value / quality functions

instead of estimating value / quality functions directly, consider parameterized models

$$V_\pi(s) \approx v(s \mid \theta) \equiv v_\theta(s) \quad \text{or} \quad Q_\pi(s,a) \approx q(s,a \mid \theta) \equiv q_\theta(s,a)$$

and estimate the parameters θ through reinforcement learning

value function approximation

there are many possible function approximators / model classes

- (deep) neural networks
- decision or regression trees
- linear feature space models
- radial basis function models
- ⋮

nowadays, it's deep neural networks all the way . . .

of particular interest in RL are the ones differentiable w.r.t. θ

- (deep) neural networks
- decision or regression trees
- linear feature space models
- radial basis function models
- $\vdots$

because ...

problem formalization

when fitting a model v_θ to a value function V_π, we may attempt to minimize the mean squared error

$$J(\theta) = \tfrac{1}{2}\,\mathbb{E}_\pi\Big[\big[V_\pi(s) - v_\theta(s)\big]^2\Big]$$

$\Leftrightarrow$ we may formalize the task of approximating V_π in terms of v_θ as the problem of solving

$$\theta_* = \operatorname*{argmin}_\theta J(\theta)$$

depending on the chosen model class for v_θ, this can be a daunting task, but ...

solution strategy

if v_θ is differentiable w.r.t. θ, we may estimate suitable parameters via gradient descent

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$

$$\Leftrightarrow \qquad \theta \leftarrow \theta + \alpha \, \mathbb{E}_\pi \Big[\Big[V_\pi(s) - v_\theta(s) \Big] \nabla v_\theta(s) \Big]$$

(we always understand ∇ to act on the parameters θ)

note

an apparent problem is that we do not know $V_\pi(s)$

note

an apparent problem is that we do not know $V_\pi(s)$

however, we may create (numerous) episodes like

$$s[0], a[0] : r[1], s[1], a[1] : \cdots : r[T], s[T], a[T]$$

and substitute empirical *estimators* such as these

the *MC target*

$$\hat{G}\big(s[t]\big) = \sum_{\tau=t}^{T-1} \gamma^{\tau-t}\, r[\tau+1]$$

the *TD target*

$$r[t+1] + \gamma\, v_\theta\big(s[t+1]\big)$$

gradient Monte Carlo optimization of $v_\theta \approx V_\pi$

initialize θ

for many, many trials
  sample a state
    $s[0] \in \mathcal{S} \setminus \mathcal{F}$

  follow policy π to generate an episode // policy π is given and independent of v_θ
    $s[0], a[0] : r[1], s[1], a[1] : \cdots : r[T], s[T], a[T]$

  for $t = 0, \ldots, T-1$
    compute

$$\hat{G}\big(s[t]\big) = \sum_{\tau=t}^{T-1} \gamma^{\tau-t}\, r[\tau+1]$$

    update

$$\theta \leftarrow \theta + \alpha \left[\hat{G}\big(s[t]\big) - v_\theta\big(s[t]\big)\right] \nabla v_\theta\big(s[t]\big)$$

claim w/o proof

the *MC target* is an *unbiased estimator* of $V_\pi(s)$

$\Leftrightarrow$ we have

$$\mathbb{E}\left[\hat{G} \mid S_0 = s\right] = V_\pi(s)$$

and are therefore guaranteed that

$$\theta \leftarrow \theta + \alpha_t \left[\hat{G} - v_\theta(S)\right] \nabla v_\theta(S)$$

converges to a local optimum of $J(\theta)$ *if* the α_t meet the Robbins-Monro conditions

claim w/o proof

the *TD target*, on the other hand, is a *biased estimator* of $V_\pi(s)$

this is because its definition involves $\gamma\, v_\theta(S_{t+1})$ and therefore the function v_θ that is to be learned

the “simple” gradient we gave above therefore doesn’t apply here

put differently, updates of the form

$$\theta \leftarrow \theta + \alpha \left[R_{t+1} + \gamma\, v_\theta(S_{t+1}) - v_\theta(S_t) \right] \nabla v_\theta(S_t)$$

would just constitute a so called semi-gradient descent procedure

note

just to be clear

the *gradient* of

$$\frac{1}{2}\left[R_{t+1} + \gamma\, v_\theta\left(S_{t+1}\right) - v_\theta\left(S_t\right)\right]^2$$

would amount to

$$\left[R_{t+1} + \gamma\, v_\theta\left(S_{t+1}\right) - v_\theta\left(S_t\right)\right]\ \left[\gamma\, \nabla\, v_\theta\left(S_{t+1}\right) - \nabla\, v_\theta\left(S_t\right)\right]$$

the above *semi-gradient* involves only half this information, hence its name

however . . .

note

everybody uses semi-gradient descent anyway

though they do not converge very robustly, they work rather well in (relevant) special cases and can learn in a model free- and online manner

semi-gradient TD(0) optimization of $v_\theta \approx V_\pi$

initialize θ

for many, many trials
  sample a state
    $s \in \mathcal{S} \setminus \mathcal{F}$

  while $s \notin \mathcal{F}$
    sample an action // policy π is given and independent of v_θ
      $a \sim \pi(\cdot \mid s)$

    get a successor // outcome depends on "physics engine"
      $s' \leftarrow Succ(s, a)$

    receive a reward // outcome depends on "physics engine"
      $r \leftarrow Util(s')$

    update
      $\theta \leftarrow \theta + \alpha \left[r + \gamma\, v_\theta(s') - v_\theta(s) \right] \nabla v_\theta(s)$

disclaimer

henceforth, we shall focus on temporal difference (TD) methods

everybody does this, because

- in practice, hardly anybody ever uses MC algorithms for RL
- the main criticism is that they must wait for episodes to end before they can learn anything

funnily enough, this point is arguably moot in what comes next ;-)

intermediate recap

we proposed to approximate $V_\pi(s) \approx v_\theta(s)$

we proposed to do this through minimizing

$$J(\theta) = \tfrac{1}{2}\,\mathbb{E}_\pi\left[\left[V_\pi(s) - v_\theta(s)\right]^2\right]$$

with

$$\nabla J(\theta) = -\mathbb{E}_\pi\left[\left[V_\pi(s) - v_\theta(s)\right]\nabla v_\theta(s)\right]$$

this looks like a supervised learning problem, but ...

inspecting our objective function and/or its gradient, we noticed that we do not / can not know the target values $V_\pi(s)$

for (understandable) practical reasons, we therefore substituted the TD target

$$Util(s') + \gamma\, v_\theta(s')$$

with this, we finally proposed to perform *semi-gradient descent*

$s \sim \mathcal{S} \setminus \mathcal{F}$
while $s \notin \mathcal{F}$
  $a \sim \pi(\cdot \mid s)$
  $s' \leftarrow Succ(s, a)$
  $\theta \leftarrow \theta + \alpha \left[Util(s') + \gamma\, v_\theta(s') - v_\theta(s) \right] \nabla v_\theta(s)$

note

experienced machine learners with a solid background in gradient descent for supervised learning would be horrified by what we just did !!!!!!!!!!!!!!!!!!!!

(*not* because we are working with semi-gradients which is generally fine)

no, what is questionable about the above semi-gradient TD(0) procedure becomes more visible upon rewriting its parameter updates as

$$\theta_{t+1} = \theta_t + \alpha \left[Util\big(s[t]\big) + \gamma\, v_{\theta_t}\big(s[t+1]\big) - v_{\theta_t}\big(s[t]\big) \right] \nabla v_{\theta_t}\big(s[t]\big)$$

now, we see two problems, namely ...

problems with episodic stochastic semi-gradient TD(0)

1) it updates the estimate of the model parameters θ of v_θ every time an episode generates a pair $\big(s[t], s[t+1]\big)$

   this is *stochastic (semi-)gradient descent* rather than true (semi-)gradient descent
   (indeed, whatever happened to the $\mathbb{E}_\pi\big[\cdots\big]$ operator in our objective $J\big(\theta\big)$ and its gradient $\nabla J\big(\theta\big)$? we tacitly ignored it, that's what!)

   however, it is common ML lore that *mini-batch gradient descent* is more robust and less noise sensitive (i.e. learns more reliably)

   moreover, episodic data are *not* i.i.d. but necessarily correlated which will further reduce the reliability of stochastic gradient descent

   finally, the episodic approach is *not sample efficient* since it never reuses previously generated data

problems with episodic stochastic semi-gradient TD(0)

2) it learns with *moving targets*

   $\Leftrightarrow$ every time it updates θ_t to θ_{t+1}, the model changes from v_{θ_t} to $v_{\theta_{t+1}}$

   $\Leftrightarrow$ at time t, the model $v_\theta(s)$ yields a different value than at time $t+1$

note

moving targets are common in control theory but not in supervised learning (once again: reinforcement learning is special)

$\Leftrightarrow$ “in general”, *moving targets* are no big deal in value function approximation (quality functions are a different story) . . .

we just must be careful when choosing learning algorithms (gradient-based optimization is “generally” fine) and hyper-parameters such as learning rates

next, we focus on problem 1)
and return to problem 2) later

batch RL / experience replay

while exploring the environment, create / update a memory $\mathcal{E}$ of experiences

S_t	A_t	R_{t+1}	S_{t+1}	*done*
$s[0]$	$a[0]$	$u(s[1])$	$s[1]$	0
$s[1]$	$a[1]$	$u(s[2])$	$s[2]$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s[h_1 - 1]$	$a[h_1 - 1]$	$u([s[h_1])$	$s[h_1]$	1
$s[h_1]$	$a[h_1]$	$u([s[h_1 + 1])$	$s[h_1 + 1]$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s[h_1 + h_2 - 1]$	$a[h_1 + h_2 - 1]$	$u(s[h_1 + h_2])$	$s[h_1 + h_2]$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

(rows are transitions together with an “end of episode flag” (for good measure))

$t \leftarrow 0$
$\mathcal{E} \leftarrow \varnothing$
$done \leftarrow TRUE$

while $t < T$
  if $done$
    $s[t] \sim \mathcal{S} \setminus \mathcal{F}$
    $done \leftarrow FALSE$
  $a[t] \sim \pi\big(\cdot \mid s[t]\big)$
  $s[t+1] \leftarrow Succ\big(s[t], a[t]\big)$
  $r[t+1] \leftarrow Util\big(s[t+1]\big)$
  if $s[t+1] \in \mathcal{F}$
    $done \leftarrow TRUE$
  add $\big(s[t], a[t], r[t+1], s[t+1], done\big)$ to $\mathcal{E}$

  if $t \mod n = 0$
    get i.i.d. sample $\big\{\big(s_i, a_i, r_i, s_i'\big)\big\}_{i=1}^{n} \subset \mathcal{E}$
    batch update model parameters $\theta \leftarrow \theta + \frac{\alpha}{n} \sum_{i=1}^{n} \Big[r_i + \gamma\, v_\theta\big(s_i'\big) - v_\theta\big(s_i\big)\Big] \nabla v_\theta\big(s_i\big)$

  $t \leftarrow t + 1$

note

the above algorithm

- reuses previously generated data
  - $\Leftrightarrow$ is generally *sample efficient*
  - $\Leftrightarrow$ works with “moderate” T
- updates in the direction of *batch gradients*
  - $\Leftrightarrow$ works with averaged gradient information
  - $\Leftrightarrow$ descends less erratic
  - $\Leftrightarrow$ learns faster
- updates from (likely more) uncorrelated data
  - $\Leftrightarrow$ descends more unbiased
  - $\Leftrightarrow$ learns more robust

linear models

note

approximate (semi-)gradients

$$\nabla J(\theta) \approx -\frac{1}{n} \sum_{i=1}^{n} \left[r_i + \gamma \, v_\theta(s_i') - v_\theta(s_i) \right] \nabla v_\theta(s_i)$$

are rather easy to compute for linear model functions

$$v_\theta(s) = \sum_{d=1}^{D} \theta_d \cdot \varphi_d(s)$$

where the $\varphi_d : \mathcal{S} \to \mathbb{R}$ are functions independent of θ

by introducing feature vectors

$$\boldsymbol{\varphi}_s = \boldsymbol{\varphi}(s) = \begin{bmatrix} \varphi_1(s) \\ \varphi_2(s) \\ \vdots \\ \varphi_D(s) \end{bmatrix}$$

we can write linear models as

$$v_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\varphi}_s$$

this "immediately" tells us that

$$\nabla v_{\boldsymbol{\theta}}(s) = \nabla \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{\varphi}_s = \boldsymbol{\varphi}_s$$

$\Rightarrow$ the general expression

$$-\frac{1}{n}\sum_{i=1}^{n}\left[r_i+\gamma\, v_\theta\left(s_i'\right)-v_\theta\left(s_i\right)\right]\nabla v_\theta\left(s_i\right)$$

considerably simplifies to

$$-\frac{1}{n}\sum_{i=1}^{n}\left[r_i+\gamma\,\theta^\top\boldsymbol{\varphi}_{s_i'}-\theta^\top\boldsymbol{\varphi}_{s_i}\right]\boldsymbol{\varphi}_{s_i}=-\frac{1}{n}\left[\underbrace{\sum_{i=1}^{n}\boldsymbol{\varphi}_{s_i}\,r_i}_{=\boldsymbol{b}\in\mathbb{R}^d}-\underbrace{\left[\sum_{i=1}^{n}\boldsymbol{\varphi}_{s_i}\boldsymbol{\varphi}_{s_i}^\top-\gamma\sum_{i=1}^{n}\boldsymbol{\varphi}_{s_i}\boldsymbol{\varphi}_{s_i'}^\top\right]}_{=\boldsymbol{A}\in\mathbb{R}^{d\times d}}\theta\right]$$

$\Rightarrow$ we have the following batch gradient

$$\nabla J(\theta) \approx \frac{1}{n}\Big[\boldsymbol{A}\theta - \boldsymbol{b}\Big]$$

at a (local) minimum θ_* of $J(\theta)$ this (empirical) gradient should be small

$$\|\boldsymbol{A}\theta_* - \boldsymbol{b}\|^2 \approx 0$$

$\Rightarrow$ estimating θ_* is but an LSQ problem

$$\theta_* = \underset{\theta}{\operatorname{argmin}} \|\boldsymbol{A}\theta - \boldsymbol{b}\|^2$$

$\Rightarrow$ we have (as $\boldsymbol{A}$ is square)

$$\theta_* = \boldsymbol{A}^{-1}\boldsymbol{b}$$

or, typically more stable

$$\theta_* = \left[\boldsymbol{A}^\intercal \boldsymbol{A}\right]^{-1}\boldsymbol{A}^\intercal \boldsymbol{b}$$

or, even more stable (*regularized*)

$$\theta_* = \left[\boldsymbol{A}^\intercal \boldsymbol{A} + \eta\, \boldsymbol{I}\right]^{-1}\boldsymbol{A}^\intercal \boldsymbol{b}$$

recall our discussion of MLE and MAP estimator from PoML

note

just to be clear

we are still working with semi-gradients

also, $\boldsymbol{A}$ and $\boldsymbol{b}$ are computed from a sample drawn from $\mathcal{E}$

$\Rightarrow$ $\frac{1}{n}\left[\boldsymbol{A}\theta - \boldsymbol{b}\right]$ is not the true gradient of the loss function $J(\theta)$

$\Rightarrow$ the above θ_* is most likely not the true minimizer of $J(\theta)$

we therefore have ...

batch RL with linear models (online)

$\vdots$

while $t < T$

$\vdots$

if $t \mod n = 0$

get $\left\{\left(s_i, a_i, r_i, s_i'\right)\right\}_{i=1}^{n} \subset \mathcal{E}$

compute $\boldsymbol{A}$, $\boldsymbol{b}$, and then $\boldsymbol{\theta}_*$

batch update model parameters $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\theta}_*$ // we still update only conservatively

$t \leftarrow t + 1$

batch RL with linear models (offline)

generate experiences $\mathcal{E}$

for N batches of size n **do**

get $\left\{\left(s_i, a_i, r_i, s_i'\right)\right\}_{i=1}^{n} \subset \mathcal{E}$

compute $\boldsymbol{A}$, $\boldsymbol{b}$, and then θ_*

batch update model parameters $\theta \leftarrow \theta + \alpha\theta_*$ // we still update only conservatively

note

the above algorithm is decidedly *not* the LSTD algorithm in Sutton & Barto, chap. 9

LSTD (*least squares TD*) does *not* perform batch learning but updates θ for each t

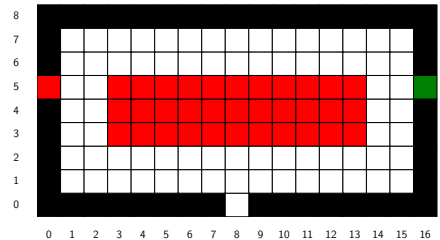
it uses the *Sherman-Morrison formula* to efficiently update $\boldsymbol{A}_t^{-1}$ which would be an interesting side quest in our overall study, however ...

we will *not* study LSTD as it simply isn't worth our time

examples

in our grid world examples, states $s_i \in \mathcal{S}$ correspond to locations

$$\boldsymbol{s_i} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \in \mathbb{N}^2$$



dealing with such states, we may consider . . .

radial basis models

assuming a set

$$\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_D\} \subset \mathbb{R}^2$$

of D *location prototypes* $\boldsymbol{p}_d$, we may model

$$v_\theta(s) = \sum_{d=1}^{D} \theta_d \cdot \phi(\boldsymbol{s}, \boldsymbol{p}_d)$$

with isotropic Gaussian radial basis kernels

$$\phi(\boldsymbol{s}, \boldsymbol{p}_d) = \exp\left(-\frac{1}{2\,\sigma^2}\|\boldsymbol{s} - \boldsymbol{p}_d\|^2\right)$$

$\Rightarrow$ letting

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_D \end{bmatrix} \quad \text{and} \quad \boldsymbol{\varphi}_s = \boldsymbol{\varphi}(s) = \begin{bmatrix} \phi(\boldsymbol{s}, \boldsymbol{p}_1) \\ \phi(\boldsymbol{s}, \boldsymbol{p}_2) \\ \vdots \\ \phi(\boldsymbol{s}, \boldsymbol{p}_D) \end{bmatrix}$$

we can write our radial basis model as an inner product

$$v_\theta(s) = \theta^\intercal \boldsymbol{\varphi}_s$$

(multi-variable) polynomials

for many good reasons we may want to normalize

$$\bar{\boldsymbol{s}} = \boldsymbol{s} - \mathbb{E}\left[\boldsymbol{s}_i \in \mathcal{S}\right] = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}$$

with this normalization, it is reasonable to consider

$$\begin{aligned} v_\theta(s) &= \sum_{k=0}^{K} \sum_{l=0}^{L} \theta_{kl}\, \bar{x}^k\, \bar{y}^l \\ &= \theta_{00}\, \bar{x}^0\, \bar{y}^0 + \theta_{01}\, \bar{x}^0\, \bar{y}^1 + \ldots + \theta_{KL}\, \bar{x}^K\, \bar{y}^L \end{aligned}$$

$\Rightarrow$ defining $D = (K+1) \cdot (L+1)$ dimensional vectors

$$\theta = \begin{bmatrix} \theta_{00} \\ \theta_{01} \\ \vdots \\ \theta_{KL} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\varphi}_s = \boldsymbol{\varphi}(s) = \begin{bmatrix} \overline{x}^0 \overline{y}^0 \\ \overline{x}^0 \overline{y}^1 \\ \vdots \\ \overline{x}^K \overline{y}^L \end{bmatrix}$$

we can write our polynomial model as an inner product

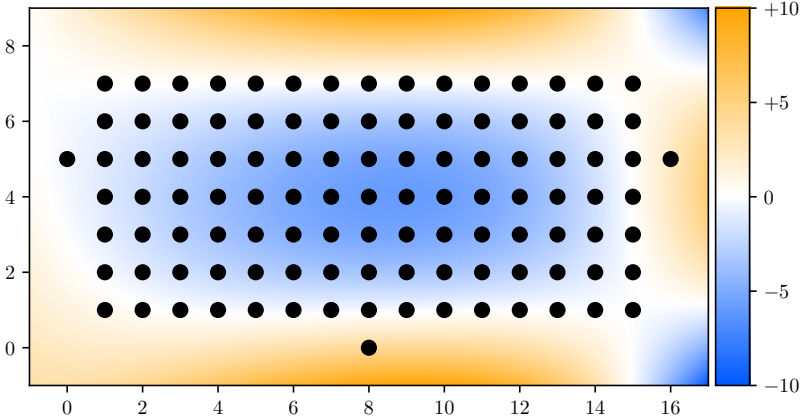
$$v_\theta(s) = \theta^\intercal \boldsymbol{\varphi}_s$$

example

$\gamma = 0.99$

$K = 3, L = 2 \Rightarrow D = 12$

$|\mathcal{E}| = 27,617, N = 10,000, n = 100, \alpha = 0.1$, and output normalization to $[-10, +10]$



remarks

given the simplicity of our polynomial model, the above **result looks good but certainly not ideal**

the latter can largely be attributed to shortcomings of TD(0) approaches for policy evaluation with value functions rather than to (our chosen model and method for) value function approximation

$\Leftrightarrow$ TD(0) targets carry not enough information

we could now try to fix this but it isn't worth our time . . . after all, we already know that value functions are not as useful as quality functions anyway

so let's move on . . .

quality function approximation

we can do all of the above with quality functions and attempt to learn

$$q_\theta(s,a) \approx Q_*(s,a)$$

just as with TD(0), we may simply consider semi-gradient updates for

SARSA

$$\theta \leftarrow \theta + \alpha \left[Util(s') + \gamma\, q_\theta(s',a') - q_\theta(s,a) \right] \nabla q_\theta(s,a)$$

Q-learning

$$\theta \leftarrow \theta + \alpha \left[Util(s') + \gamma \max_{a'} q_\theta(s',a') - q_\theta(s,a) \right] \nabla q_\theta(s,a)$$

note: the Q-learning updates make it clear why RL people prefer semi-gradients

note

however, when approximating quality functions, we must carefully consider what kind of models $q_\theta(s, a)$ to work with

this is because states s and actions a or (informed) feature vectors φ_s or ψ_a can be like apples and oranges and (naïve) gradient descent over functions of apples and oranges may not behave as intended

practitioners therefore prefer to model quality functions with neural networks whose weights θ can be learned end-to-end without much worry about input normalization and the like ...

note

moreover, in quality function approximation with SARSA or Q-learning targets, the issue of moving targets becomes a concern as successor states depend on actions sampled ϵ-greedily w.r.t. q_θ

$\Leftrightarrow$ any update of the model parameters not only changes the model but implicitly also the policy used for generating experiences

in the approximative setting (*on-* or *off-policy*), this may cause harm w.r.t. robustness and convergence behavior of the learning process (unwanted feedback loops, oscillations, . . .)

$\Leftrightarrow$ we could spend several lectures on the numerous workarounds that have been developed to address these and other intricacies (or read chapters 10 and 11 in Sutton & Barto)

however . . .

note

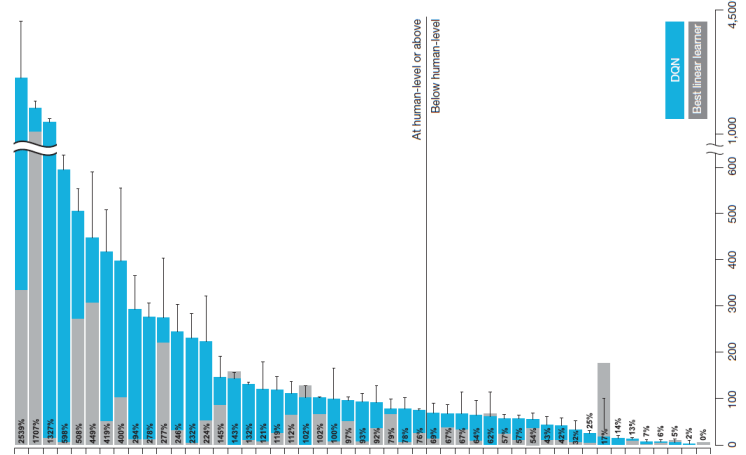
in 2015, Google Deepmind shook things up when they introduced deep Q-learning

> V. Mnih et al., *Human-level Control through Deep Reinforcement Learning*, Nature, 518(7540), **2015**

their system learned control policies $\pi(a \mid s)$ for classic ATARI arcade games where actions consisted of 18 joystick positions (small and discrete set $\mathcal{A}$) and states were screenshots / pixels images (arbitrarily large sets $\mathcal{S}$)

what shocked the world back then was that they were able to learn from rendered game states (pictures) rather than from compact internal code representations

results


Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
469%
419%
400%
294%
278%
277%
246%
232%
224%
145%
142%
132%
121%
113%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
55%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
0
100
200
300
400
500
600
1,000
4,500%
DQN
Best linear learner

observe

their agents show superhuman performance for simple reactive games (*pinball*, *breakout*, . . .) but fail miserably in games that require more strategic behaviors (*Montezuma's revenge*, . . .)

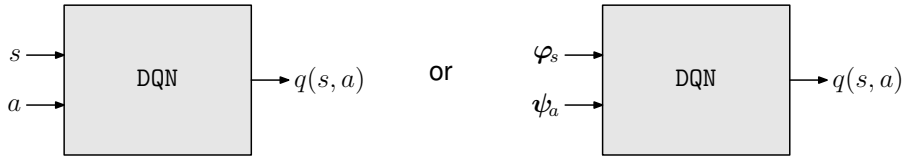
questions

be that as it may, learning from pixels is impressive . . . how did they do it ?

answer

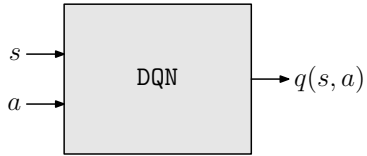
well . . .

first of all, they observed that networks like these

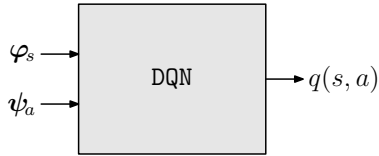


would require $|\mathcal{A}| = m$ forward passes to determine an appropriate action a given s

first of all, they observed that networks like these

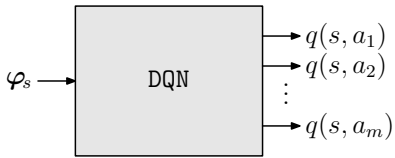


or



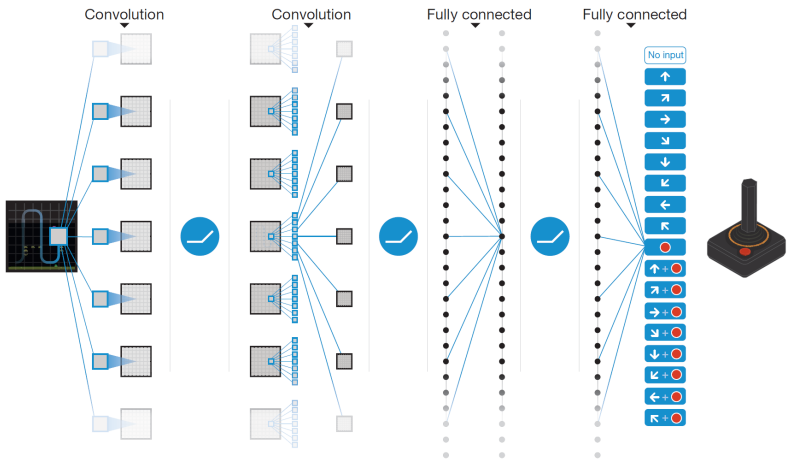
would require $|\mathcal{A}| = m$ forward passes to determine an appropriate action a given s

breaking with tradition, they therefore opted for Q-learning with networks like this



more specifically ...

convolutional DQN



second of all, they learned with experience replay

third of all and most importantly, they worked with two models q_{θ_b} and q_{θ_e} to address the *moving target problem* and thus to learn more successfully

> how did they come up with this? likely through an engineering-driven root cause analysis of why a single DQN didn't learn anything ;-)

all in all, their *deep Q-learning with experience replay* algorithm reads ...

$\mathcal{E} \leftarrow \varnothing$

initialize behavior DQN q_{θ_b} with weights θ_b randomly
initialize estimate DQN q_{θ_e} with weights $\theta_e = \theta_b$

for N episodes **do**
  sample $s \in \mathcal{S} \setminus \mathcal{F}$
  while $s \notin \mathcal{F}$
    sample $a \in \mathcal{A}(s)$ ϵ-greedily w.r.t. q_{θ_b} // behavior policy depends on q_{θ_b}
    $s' \leftarrow Succ(s, a)$
    $r \leftarrow Util(s')$
    add (s, a, r, s') to $\mathcal{E}$

    get i.i.d. sample $\{(s_i, a_i, r_i, s_i')\}_{i=1}^{n} \subset \mathcal{E}$
    set $y_i \leftarrow \begin{cases} r_i & \text{if } s_i' \in \mathcal{F} \\ r_i + \gamma \max\limits_{a'} q_{\theta_e}(s', a') & \text{otherwise} \end{cases}$ // learning targets depend on q_{θ_e}
    update θ_b trough backpropagation on $\sum_i \left[y_i - q_{\theta_b}(s_i, a_i) \right]^2$ // use, say, RMSProp, Adam, ...

    every c steps, reset $\theta_e \leftarrow \theta_b$

note

$\Leftrightarrow$ every c steps, the target network q_{Θ_e} is cloned from the behavior network q_{Θ_b}

"This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $q(s, a)$ often also increases $q(s', a)$ for all a and hence also increases the target y_i, possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time an update to q is made and the time the update affects the targets y_i, making divergence or oscillations much more unlikely."

and then there is

"We also found it helpful to clip the error term from the update $r + \gamma \max_{a'} q_\Theta(s, a') - q_\Theta(s, a)$ to between -1 and 1. Because the absolute value loss function $|x|$ has a derivative of -1 for all negative values of x and a derivative of 1 for all positive values of x, clipping the squared error to be between -1 and 1 corresponds to using an absolute value loss function for errors outside of the $(-1, 1)$ interval. This form of error clipping further improved the stability of the algorithm."

cliffhanger

questions

why consider approximations of $V_\pi(s)$ or of $Q_\pi(s,a)$?

why not directly consider approximations of $\pi(a \mid s)$?

answer

why indeed ?

summary

we now know about

reinforcement learning in settings with overly large state- and/or action spaces

value- and quality-function approximations in terms of parameterized models

$$V_\pi(s) \approx v(s \mid \theta) \equiv v_\theta(s) \qquad \text{and} \qquad Q_\pi(s,a) \approx q(s,a \mid \theta) \equiv q_\theta(s,a)$$

and the the training of differentiable such models via (semi-)gradient descent

the basic ideas behind deep Q-learning