ELEC96033: Deep Learning

Carlo Ciliberto and Krystian Mikolajczyk

Department of Electrical and Electronic Engineering
Imperial College London

Previously

We studied:

- **Supervised Learning:** we are given x and y from $P(x, y)$, we want to learn $f : x \mapsto y$ approximating $P(y|x)$.
- **Unsupervised Learning:** we are given x we want to approximate $P(x)$

Today we focus on **Reinforcement Learning**:

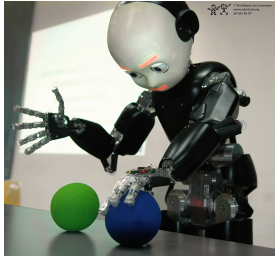
- We are not given examples of x or y, we are allowed to choose how to sample them.

Recap: What is Reinforcement Learning?

Reinforcement Learning (RL) is what one usually think of when speaking about AI:

Learning from:

- Experience,
- Exploration,
- Interaction,
- Observing cause-effect relations.



Setting: an agent (e.g. a robot) interacts with the environment in order to carry out a task.

Examples


nature
LEARNING CURVE
nature
At last — a computer program that can beat a champion Go player
ALL SYSTEMS GO


DEEP
BLUE

Recap: What is Reinforcement Learning?

Key aspects:

- Typically, the task cannot be solved with a single action (we have a dynamic problem),
- In the beginning, we (usually) don't know the effect of an action on the environment,

Two joint (and often simultaneous) learning problems:

- learn how the agent's actions affect the environment,
- how to use those actions to solve the task!

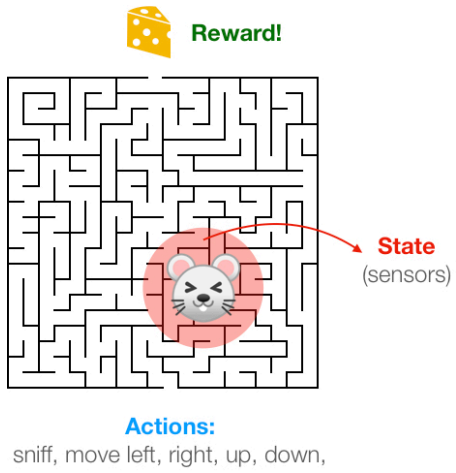
RL: Main Elements

The RL process is composed by a succession of:

- **(States)** Experience/Sense the current state of the world,
- **(Actions)** Act upon such perceptual information,
- **(Rewards)** Receive feedback following last action,
- wash, rinse, repeat...

Goal: maximise the reward!

RL: Main Elements



RL: States, Actions and Rewards

We denote by:

- $\mathcal{S}$ the set of *states* (for now assume finite cardinality),
- $\mathcal{A}$ the set of *actions* (for now assume finite cardinality),
- we assume *rewards* to be scalar values in $\mathbb{R}$ (typically bounded).

Markov Decision Process

We consider two probability distributions:

- **Transition Probability:** $P(r, s'|s, a)$, the probability of getting reward r and transitioning to the new state s' when performing action a from state s.
- **Policy:** $\pi(a|s)$ the probability that the agent will perform action a when it is in state s.

Definition. The triplet $(\mathcal{S}, \mathcal{A}, P)$ is called a Markov Decision Process (MDP).

Note. the term Markov follows from the dependency of the new state and reward on only the previous state and action (and not on also older ones).

Markov Decision Process

Note:

- $P(r, s'|s, a)$ is how the environment "works". The agent does not have control over it (and usually does not "know" it).
- $\pi(a|s)$ is the behavior of the agent. Our goal is to change it to optimize the overall reward accumulated (return).

So what do we want to maximise exactly?

Goal of RL: Maximize the Return

Consider a run of the MDP starting from a state $s = s_0$ with policy π for T iterations...

$$s = s_0, a_0, r_1, s_1, a_1, \ldots, r_T, s_T.$$

This is called an episode.

Return: the reward accumulated over one run of the MDP from

$$g(s) = \sum_{t=1}^{T} r_t$$

Goal: find π_* that maximises the expected return $\mathbb{E}\, g(s)$ (possibly for any starting point s).

Discounted Returns

What if:

- We do not know when the agent is going to stop?
- ... or the agent could go on indefinitely?

We would have infinitely long episodes ($T = +\infty$), with possibly unbounded return.

We consider

$$g(s) = \sum_{t=1}^{+\infty} \gamma^{t-1} r_t$$

where $\gamma \in [0, 1)$ is a discounting factor.

Intuitively: γ represents how the agent aims to be efficient in long-term.

Value Function

Given a policy π, we can define its **Value function** in a state s

$$V_\pi(s) = \mathbb{E}\, g(s) = \mathbb{E}\left[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s_0 = s\right]$$

Namely the expected return when following the policy π from s.

We can therefore formally state the RL learning problem as that of finding

$$\pi_* = \underset{\pi}{\text{argmax}}\, V_\pi(\cdot)$$

Question: does such policy exist?

Answer: yes!

Action-Value Function

We can also define the **Action-Value function (or Q-function)**:

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s, a\right] = \mathbb{E}[r_1|s,a] + \gamma \mathbb{E}[v_\pi(s')|s,a]$$

The return expected by first performing action a from state s and then following the policy π.

Note: if π is deterministic, namely we have $\pi(a|s) = \delta_{\bar{\pi}(s)}(a)$ with $\bar{\pi} : \mathcal{S} \to \mathcal{A}$ a function. Then

$$Q_\pi(s, \bar{\pi}(s)) = V_\pi(s)$$

(for simplicity we denote $\bar{\pi}$ as simply π when is clear from the context)

Greedy Policy

Why is the Action-Value function useful when we already have V?

Because we can use it to improve on a deterministic policy:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}} \; Q_\pi(s, a)$$

Then we have

$$V_{\pi'}(s) = Q_{\pi'}(s, \pi'(s)) \geqslant Q_\pi(s, \pi(s)) = V_\pi(s)$$

Policy Iteration

This suggest a strategy to find the best policy. Start with π_0 and then improve it "greeedily"

$$\pi_0 \xrightarrow{greedy} \pi_1 \xrightarrow{greedy} \pi_2 \to \cdots \to \pi_t \to \ldots$$

Question: Does this converge (hopefully to π_*)? **Yes!**

Problem: We need to know to Q_{π_t} for any policy π_t!

RL and Dynamic Programming

In practice: the agent does not know how $P(r, s'|a, s)$ works...

... but what if it did? Would it be able to find a good (well optimal really) policy?

Dynamic Programming:

Assume perfect knowledge of $P(r, s'|a, s)$.

How do we use this to estimate Q_π?

Bellman Equation

By definition of Q_π, for any state-action pair (s, a),

$$Q_\pi(s, a) = \mathbb{E}[r_1|s, a] + \gamma \mathbb{E}[Q_\pi(s', \pi(s'))|s, a]$$

This is known as Bellman's Equation for the action-value function.

How is this useful? Well... consider T_π the Bellman operator sending functions to functions, such that, for any $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,

$$(T_\pi f)(s, a) = \mathbb{E}[r_1|s, a] + \gamma \mathbb{E}[f(s', \pi(s')|s, a]$$

Then, by Bellman's equation $Q_\pi = T_\pi Q_\pi$ is a fixed point of T_π!

The Bellman Operator is a Contraction

Sometimes, fixed points are easy to find. In particular if the operator is a contraction!

For any $f, g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we have

$$\begin{aligned}
\|T_\pi f - T_\pi g\|_\infty &= \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} |(T_\pi f)(s,a) - (T_\pi g)(s,a)| \\
&= \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \gamma \left| \mathbb{E}\Big[f(s', \pi(s')) - g(s', \pi(s')) \Big| s, a \Big] \right| \\
&\leqslant \gamma \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} |f(s,a) - g(s,a)| = \gamma \|f - g\|_\infty
\end{aligned}$$

Since $0 \leqslant \gamma < 1$ this implies that T_π is a contraction!

Fixed Point Iteration for Contractions

It is easy to find the fixed points of a contraction. Just apply the operator many times to a starting point f_0!

$$\begin{aligned}\left\| T_\pi^k f_0 - Q_\pi \right\|_\infty &= \left\| T_\pi^k f - T_\pi Q_\pi \right\|_\infty \\ &\leqslant \gamma \left\| T_\pi^{k-1} f_0 - Q_\pi \right\|_\infty \leqslant \cdots \leqslant \gamma^k \left\| f_0 - Q_\pi \right\|_\infty\end{aligned}$$

Clearly $\lim_{k \to +\infty} T_\pi^k f_0 = Q_\pi$.

...and we also have a rate of convergence with respect to the number of iterations k!

Dynamic Programming

So we can now do the following:

Initialize: with any policy π_0,

For $t = 0, \ldots, T, \ldots$

- find $\hat{Q}_{\pi_t}$ by
  - Initialize a $f_0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
  - Apply T_π for k times to f_0, obtaining $\hat{Q}_{\pi_t} = T_{\pi_t}^k f_0 \approx Q_{\pi_t}$
- Greedily find π_{t+1} as $\text{argmax}_{a \in \mathcal{A}} \hat{Q}_{\pi_t}(s, a)$.

This works! And converges to to the optimal solution!

Back to RL

Ok, what does this teach us? That if we know the Bellman operators we are in a good shape!

In practice (again): we do not know P and thus we cannot access T_π

But we have examples of how P acts from individual episodes (experience)! Can we use them?

Remember:

- we do not need direct knowledge of P but rather of Q_π.
- Q_π is defined in terms of expectations $Q_\pi(s,a) = \mathbb{E}[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s, a]$
- We know that the expectation can be well approximated by empirical averages!

This suggests a potential approach to approximate Q_π stochastically...

Monte Carlo (MC) Sampling

Let's collect many episodes...

$$
\begin{aligned}
&s_0^{(1)}, a_0^{(1)}, r_1^{(1)}, \ldots, s_t^{(1)}, a_t^{(1)}, r_{t+1}^{(1)}, \ldots \\
&s_0^{(2)}, a_0^{(2)}, r_1^{(2)}, \ldots, s_t^{(2)}, a_t^{(2)}, r_{t+1}^{(2)}, \ldots \\
&\vdots \\
&s_0^{(n)}, a_0^{(n)}, r_1^{(n)}, \ldots, s_t^{(n)}, a_t^{(n)}, r_{t+1}^{(n)}, \ldots
\end{aligned}
$$

We can build a "dataset" of the form $(s_i, a_i, g_i)_{i=1}^N$, where

- s_i and a_i are the states and actions visited during the episodes (e.g. $s_i = s_t^{(j)}$ and $a_i = a_t^{(j)}$ for episode j).
- g_i is the corresponding return for $(s, a) = (s_i, a_i)$ for that episode (e.g. $g_i = \sum_{\ell=1}^{T} \gamma^{\ell-1} r_\ell^{(j)}$).

Monte Carlo Sampling (Cont.)

For any state action pair (s, a) we define

- $N(s, a) = \sum_{i=1}^{N} \delta_{s,a}(s_i, a_i)$. The number of times (s, a) was visited,
- $G(s, a) = \sum_{i=1}^{N} g_i \delta_{s,a}(s_i, a_i)$. The cumulative return obtained when visiting (s, a) and completing the episode.

We can therefore obtain our estimate of Q_π as

$$\hat{Q}_\pi(s, a) = \frac{G(s, a)}{N(s, a)}$$

Under "mild" assumptions $\mathbb{E}\ \hat{Q}_\pi(s, a) = Q_\pi(s, a)$!

Monte Carlo Sampling + Policy Iteration

So we can now do the following:

Initialize: with any policy π_0,

For $t = 0, \ldots, T, \ldots$

- find $\hat{Q}_{\pi_t}$ by
  - Collecting state-action-returns from many episodes,
  - Incrementally updating $\hat{Q}_{\pi_t}$ by taking empirical averages.
- Greedily find π_{t+1} as $\text{argmax}_{a \in \mathcal{A}} \hat{Q}_{\pi_t}(s, a)$.

But we need to wait and collect many episodes before updating $\hat{Q}_{\pi}$...

Monte Carlo Sampling: Incremental Perspective

If we are impatient and cannot wait to have seen n episodes we could...

For every new completed episode $s_0, a_0, r_1, \ldots, s_T, a_T, r_{T+1}$

Every time we have visited $(s, a) = (s_t, a_t)$ with return g_t, we update:

- $N(s, a) = N(s, a) + 1$
- $\hat{Q}_\pi(s, a) = \hat{Q}_\pi(s, a) + \frac{1}{N(s,a)}(g_t - \hat{Q}_\pi(s, a))$

It looks a lot like a gradient descent step! We could replace $\frac{1}{N(s,a)}$ with a "step-size" α

$$\hat{Q}_\pi(s, a) = \hat{Q}_\pi(s, a) + \alpha(g_t - \hat{Q}_\pi(s, a))$$

Monte Carlo Sampling + Policy Iteration

So we can now do the following:

Initialize: with any policy π_0,

For $t = 0, \ldots, T, \ldots$

- find $\hat{Q}_{\pi_t}$ by
  - For each new state-action-return sequence episode,
    - ⋆ Update $\hat{Q}_{\pi_t}$ via incremental averaging.
- Greedily find π_{t+1} as $\text{argmax}_{a \in \mathcal{A}} \hat{Q}_{\pi_t}(s, a)$.

But we still need to wait the end of each episode to update $\hat{Q}_{\pi}$...

Temporal Differences (TD)

If we are still impatient we would like to avoid waiting the end of the episode...

One way of doing this is by recalling that

$$Q_\pi(s,a) = \mathbb{E}[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s, a] = \mathbb{E}[r_1|s,a] + \gamma \mathbb{E}[Q_\pi(s', \pi(s'))|s,a]$$

So far, we have used empirical averages of the g_t to approximate $\mathbb{E}[\sum_{t=1}^{+\infty} \gamma^{t-1} r_t | s, a]$...

What if we use empirical averages to approximate $\mathbb{E}[r_1|s,a] + \gamma \mathbb{E}[Q_\pi(s', \pi(s'))|s,a]$?

We get the Temporal Differences approach...

Temporal Differences (TD) (Cont.)

Given an episode $s_0, a_0, r_1, \ldots, s_T, a_T, r_{T+1}$

Instead of performing Monte Carlo updates

$$\hat{Q}_\pi(s,a) = \hat{Q}_\pi(s,a) + \alpha(g_t - \hat{Q}_\pi(s,a))$$

We approximate $\mathbb{E}[r_1|s,a] + \gamma\mathbb{E}[Q_\pi(s', \pi(s'))|s,a]$ with the "sample" $r_t + \hat{Q}_\pi(s_{t+1}, a_{t+1})$

$$\hat{Q}_\pi(s,a) = \hat{Q}_\pi(s,a) + \alpha(r_t + \hat{Q}_\pi(s_{t+1}, a_{t+1}) - \hat{Q}_\pi(s,a))$$

Note: we do not need to wait the end of the episode. We can just update $\hat{Q}_\pi$ as we go!

Monte Carlo Sampling or Temporal Differences + Policy Iteration

So we can now do the following:

Initialize: with any policy π_0,

For $t = 0, \ldots, T, \ldots$

- find $\hat{Q}_{\pi_t}$ by
  - Collecting state-action-returns from many episodes,
  - Incrementally updating $\hat{Q}_{\pi_t}$ with MC or TD.
- Greedily find π_{t+1} as $\text{argmax}_{a \in \mathcal{A}} \hat{Q}_{\pi_t}(s, a)$.

SARSA

Both TD and MC approaches need to wait until $\hat{Q}_\pi$ is a sufficiently good approximation of Q_π before updating the policy $\pi \xrightarrow{greedy} \pi'$.

What if we are *very* impatient?

We could try to jointly:

- approximate Q_π and,
- continuously update the previous policy to be greedy with respect to the current $\hat{Q}$.

In particular: given a function $\hat{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (initialized at random), we always follow the greedy policy according to $\hat{Q}$, while updating it according to the TD strategy.

SARSA (Cont.)

Initialize: from any $\hat{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

For every episode

- Start from s_0 (e.g. sampled randomly)
- Choose $a_0 = \text{argmax}_{a \in \mathcal{A}} \; Q(s_0, a)$
- For every step $t = 0 \ldots, T - 1$
  - Observe the reward r_{t+1} and new state s_{t+1},
  - Choose $a_{t+1} = \text{argmax}_{a \in \mathcal{A}} \; Q(s_{t+1}, a)$
  - Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

This is... SARSA: name comes from the fact that updates of Q depend on (s, a, r, s', a')

SARSA and Greedy Strategies (Cont.)

Too Greedy: if for $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ we follow the greedy strategy

$$\pi_Q(s) = \operatorname*{argmin}_{a \in \mathcal{A}} \; Q(s, a)$$

we risk never visiting potentially important states (not enough exploration!).

Epsilon Greedy: given a $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we define the ε-greedy policy as

$$\pi_{Q,\varepsilon}(a|s) = (1 - \varepsilon) \; \delta_{\pi_Q(s)}(a) + \varepsilon \; \mathrm{Unif}(\mathcal{A})$$

namely the policy choosing:

- the greedy action with probability $(1 - \varepsilon)$,
- a random action with probability ε

SARSA and (ε) Greedy Strategies

Initialize: from a any $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

For every episode

- Start from s_0 (e.g. sampled randomly),
- Sample a_0 from $\pi_{Q,\varepsilon}(a|s_0)$,
- For every step $t = 0 \ldots, T-1$
  - Observe the reward r_{t+1} and new state s_{t+1},
  - Sample a_{t+1} from $\pi_{Q,\varepsilon}(a|s_{t+1})$,
  - Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

We can have ε decay to zero as we iterate to asymptotically converge to a deterministic policy.

On-Policy and Off-Policy RL Methods

SARSA is a so-called on-policy method: we explore using the same policy we improve.

We can consider off-policy methods: learn the optimal policy while using an "exploratory" one

This could be useful to:

- Learn from experts (imitation learning / inverse RL)
- Re-use data from previous policies

A well-established example is Q-Learning. Requires a small change to SARSA.

Q-Learning

For a given $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ we consider

- The target policy to be π_Q: deterministic and greedy with respect to Q.
- The exploratory policy to be $\pi_{Q,\varepsilon}$: ε-greedy with respect to Q.

At every iteration starting from a state s, we:

- Sample the new action a according to the exploratory policy $\pi_{Q,\varepsilon}$ (we want to explore)
- Get reward r and transition into a new state s',
- Update Q according to the behavior of the target (greedy) policy π_Q

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a))$$

Q-Learning (Cont.)

Initialize: from a any $\hat{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

For every episode

- Start from s_0 (e.g. sampled randomly),
- For every step $t = 0 \ldots, T - 1$
  - Sample a_0 from $\pi_{Q,\varepsilon}(a|s_0)$,
  - Observe the reward r_{t+1} and new state s_{t+1},
  - Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t))$

We do not need ε to decay to zero as we iterate since π_Q is our target.

Infinite States

So far we have implicitly assumed the state space $\mathcal{S}$ to have finite cardinality

What if:

- $\mathcal{S}$ has extremely large cardinality (e.g. space of all natural images)?
- $\mathcal{S}$ has infinite cardinality (e.g. $\mathcal{S} = \mathbb{R}^d$)?

Well... we can use a function approximator:

$$Q_\pi(s, a) \approx Q(s, a|\theta)$$

where θ is a set of parameters.

RL with Function Approximators

Given a policy π, consider $Q(s, a|\theta)$ to be a neural network.

We would like find $\hat{\theta}$ such that $Q_\pi(s, a) \approx Q(s, a, \theta)$. A natural approach is to solve:

$$\hat{\theta} = \underset{\theta}{\text{argmin}} \ \mathbb{E}_\pi(Q_\pi(s, a) - Q(s, a, \theta))^2$$

SGD: we sample $(s_t, a_t, r_{t+1}, s_{t+1})$ according to π and perform a descent step

$$\theta = \theta + \alpha \ (Q_\pi(s, a) - Q(s, a|\theta)) \ \nabla_\theta Q(s, a|\theta)$$

As usual: We do not know $Q_\pi(s, a)$. So we substitute with $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}|\theta)$

RL with Function Approximators (Cont.)

We now have

$$\theta = \theta + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}|\theta) - Q(s, a|\theta))\ \nabla_\theta Q(s, a|\theta)$$

Does it reminds us of something? Well...

$$\hat{Q}(s_t, a_t) = \hat{Q}(s_t, a_t) + \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t))$$

Was the update of TD learning!

We have just described Temporal Differences (TD) with function approximation!

Q-Learning with Function Approximators

Following the same strategy we can recover all the methods described so far.

For instance, let us consider **Q-Learning** with function approximation:

Initialize: any set of network parameters θ.

For every episode:

- Start from s_0 (e.g. sampled randomly),
- For every step $t = 0 \dots, T-1$
  - Sample a_0 from $\pi_{Q,\varepsilon}(a|s_0)$,
  - Observe the reward r_{t+1} and new state s_{t+1},
  - Update $\theta = \theta + \alpha(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a'|\theta) - Q(s_t, a_t|\theta))\ \nabla_\theta Q(s_t, a_t|\theta)$

Going further: Experience Replay

We can do many things to improve the performance. For example...

Replay Memory: a set $\mathcal{D}$ collecting previous experiences as samples (s, a, r, s')

Every time we update $Q(\cdot, \cdot|\theta)$ we:

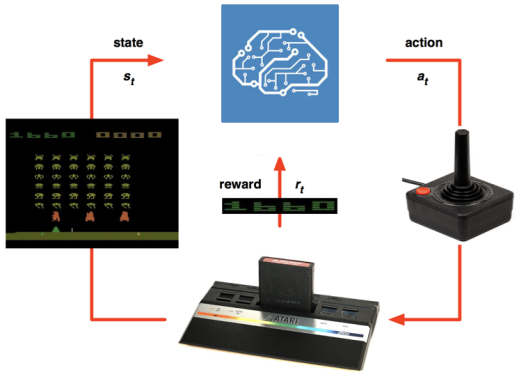
- Sample a mini-batch $(s_i, a_i, r_i, s_i')_{i=1}^n$ from $\mathcal{D}$
- Perform a descent step with gradient

$$\sum_{i=1}^{n} r_i + \gamma \max_{a' \in \mathcal{A}} Q(s_i', a'|\theta) - Q(s_i, a_i|\theta)) \ \nabla_\theta Q(s_i, a_i|\theta)$$

Example: Atari Games

Goal: play video games as humans do:

- State: Images (the raw pixels!)
- Actions: Joystick actions (e.g. left, rigth, etc.)
- Reward: Current score.



Example: Atari Games with Deep Q-Learning (DQL)

Network model for $Q(s, a|\theta)$:

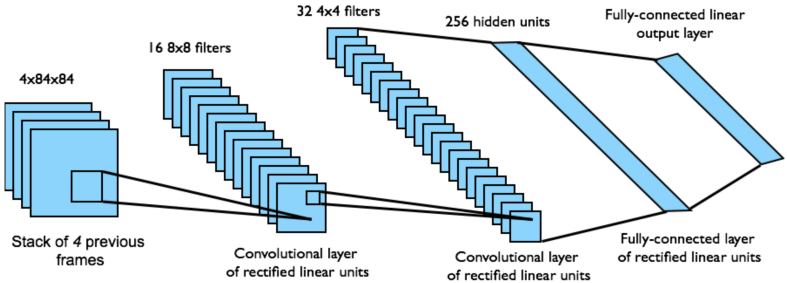


Image credits: David Silver

States: the last four frames are collated into a single state

(Accounting for potential non-Markovianity of single frames)

Example: Atari Game Video

Video

Summary

This Class. We have reviewed the main ideas behind RL and MDPs. In particular:

- The key role of Action-Value (or Q) Functions
- How to leverage complete knowledge of the MDP to find an optimal policy.
- Use empirical observations to approximate an unknown MDP.
- Deep Learning as a way to approximate the Q-functions while finding the optimal policy.

Next class. We will delve deeper in the realm of deep reinforcement learning. We will consider policy optimization: rather than using DL to approximate Q, we will use it to parametrize $\pi = \pi_\theta$ and optimize with respect to θ!