Machine Learning

Reinforcement Learning

Dr Chris Willcocks

Department of Computer Science
Recap

- GANs, AEs, Latent spaces, UNet....
  - Mostly differentiable problems up until now
- Synaptic plasticity

Today's lecture: Reinforcement Learning

"Reinforcement"

- "The action or process of reinforcing or strengthening"
- "The process of encouraging or establishing a belief or pattern of behaviour"

Further Reading: Reinforcement Learning: An Introduction  Lilian Weng David Silver
(if you're interested in learning more about this field)
Given an agent in an unknown environment, learn how to take **smart actions** such as to **maximize cumulative rewards**

- There is **no supervisory signal**, only a notion of reward $R_t$ at step $t$
- Feedback is **not immediate**
- **Time** is important (sequential, data is not IID)
- Agents actions affect subsequent environment state data
Examples

Control behaviour of complex systems

- Defeat champion at Go
- Do stunt manoeuvres in drones
- Beat humans at video games
- Find most efficient robot walking strategy over complex terrain
- Control power stations
- Self-driving cars
- Robotic cook dinner
- Investing in stock market
- ...
Learning to drive and flip pancakes

Wayve: Learning to drive in a day

Learning to flip pancakes
Interaction between **agent** and **environment** through sequential time steps $t = 1, 2, \ldots, T$. The agent **learns** about the environment, it learns the optimal **policy** and makes decisions about how to choose the next action.

Interaction sequence is described by an **episode** (trajectory) consisting of **states**, **actions**, and **rewards** at given time steps: $S_t, A_t$ and $R_t$:

$$S_1, A_1, R_2, S_2, A_2, \ldots, S_T$$

There are many categories of RL algorithm, generally they are:

- **Model-based**: The agent learns a model of the environment
- **Model-free**: The agent learns which actions to take without an environment model
- **On-policy**: Learning on the job
- **Off-policy**: Learning optimal policy independent of current policy being executed
Key Concepts
Modelling the environment’s transition and reward

The **model** describes the environment

1. Transition probability function $P$
2. Reward function $R$

Transition steps: $<s, a, s', r>$

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a)$$

Key Concepts

Policy and value

- The agent’s “policy” $\pi$ describes which action it should take in a given state.

  It can be deterministic: $a = \pi(s)$

  or stochastic:

  $$a = \pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

- The value function estimates the future reward for a state or an action. The future reward is called the return:

  $$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$
The value function estimates the future reward for a state or an action.

The state-value for a state \( s \) is the expected return at time \( t \):

\[
V_\pi (s) = \mathbb{E}_\pi [G_t | S_t = s]
\]

The action-value (Q-value, or “quality”) for a state-action pair is similarly:

\[
Q_\pi (s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a]
\]

The advantage is the difference between the action-value and state-value:

\[
A_\pi (s, a) = Q_\pi (s, a) - V_\pi (s)
\]
The **optimal value** function $V_*(s)$ produces the maximum return

$$V_*(s) = \max_\pi V_\pi(s), \quad Q_*(s, a) = \max_\pi Q_\pi(s, a)$$

The **optimal policy** $\pi_*$ is the policy that corresponds to the optimal value function

$$\pi_* = \arg \max_\pi V_\pi(s), \quad \pi_* = \arg \max_\pi Q_\pi(s, a)$$

where $V_{\pi_*}(s) = V_*(s)$ and $Q_{\pi_*}(s, a) = Q_*(s, a)$
Most reinforcement models as **Markov decision processes** $\mathcal{M} = < S, A, P, R, \gamma >$

1. Set of states $S$
2. Set of actions $A$
3. Transition function $P(s'|s, a)$
4. Reward function $R(s, a, s')$
5. Discount factor $\gamma \in [0, 1]$

- Initial state $s_0$
- Discount factor $\gamma \in [0, 1]$ for return $G_t$

$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

*from David Silver's slides*
Markov States

“The future is independent of the past given the present”

A state $S_t$ is Markov if and only if

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, ..., S_t]$$

- The state captures all information from the history
- Once the state is known, the history can be thrown away
- The state is a sufficient statistic of the future
How to make decisions?

- Select actions to *maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- Sometimes sacrifice of immediate reward is necessary for long-term reward
  - Refuelling a racing car

Each step $t$

1. **Agent**
   a. Executes $A_t$
   b. Receives observation $O_t$
   c. Receives reward $R_t$

2. **Environment**
   a. Receives $A_t$
   b. Emits observation $O_{t+1}$
   c. Emits reward $R_{t+1}$

*Partial observability*

*from slides by David Silver*
Common Approaches

- Dynamic Programming
- Monte-Carlo methods
- Temporal-Difference Learning
- Policy Gradients
- Evolution Strategies

http://blog.otoro.net/2017/10/29/visual-evolution-strategies/
If we have complete information about the environment, and know $P(s'|s, a)$ and $R(s, a, s')$, we can use Dynamic Programming to directly solve MDP's by applying the Bellman Optimality Equations, which show that:

$$V(s) = \mathbb{E}[G_t|S_t = s]$$
$$= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$$

We can then iteratively evaluate the value function and improve the policy:

$$\pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{refine}} \pi_1 \xrightarrow{\text{eval}} V_{\pi_1} \xrightarrow{\text{refine}} \pi_2 \xrightarrow{\text{eval}} \ldots \xrightarrow{\text{refine}} \pi^* \xrightarrow{\text{eval}} V^*$$
Monte-Carlo (MC) methods simply learn through experience without modelling the environment dynamics.

Typical approach:

1. Improve the policy greedily with respect to current value function
   \[ \pi(s) = \arg\max_{a \in A} Q(s, a) \]

2. Generate new trajectory with new policy \( \pi \) with \( \epsilon \)-greedy policy.

3. Estimate the new state-action return \( Q \), where
   \[
   q_\pi(s, a) = \frac{\sum_{t=1}^{T} (1[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1})}{\sum_{t=1}^{T} 1[S_t = s, A_t = a]}
   \]
Temporal-Difference (TD) learning is where we slowly move the value function $V(S_t)$ towards $R_{t+1} + \gamma V(S_{t+1})$ which is the TD target by some learning rate hyper parameter $\alpha$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- No reliance on rewards
- No complete returns (incomplete episodes/trajectories)

**Bootstrapping** update targets with regard to existing estimates, rather than relying on actual rewards and complete returns.
The previous slide also applies for estimating the return of state-action pairs:

\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \]

But what if we did an off-policy approach, instead of sampling SARSA with \( \varepsilon \)-greedy, estimating \( Q^* \) out of the best values independent of the current policy?

\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)) \]

Memorizing \( Q^*(.) \) for all state-action pairs is large/expensive. So we use a function approximator \( Q(s, a; \theta) \) aka a Deep Neural Network.

\[
\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2 \right]
\]
Common Approaches

TD Learning: **Deep Q-Learning**

---

**Algorithm 1 Deep Q-learning with Experience Replay**

- Initialize replay memory $\mathcal{D}$ to capacity $N$
- Initialize action-value function $Q$ with random weights

for episode $= 1, M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ do
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

---

**Achievement unlocked!** You can now play Atari games!
Common Approaches

Policy Gradients

Previously, we've tried to learn the state/action value and choose actions.

What if we want to learn the policy function $\pi(a|s; \theta)$ with respect to some parameters $\theta$?

How would we train this?
First, we define the reward function

\[ \mathcal{J}(\theta) = V_{\pi_\theta}(S_1) = \mathbb{E}_{\pi_\theta}[V_1] \]

To do gradient ascent, we need the partial derivative of the reward function with respect to the parameters, e.g. *numerically*

\[ \frac{\partial \mathcal{J}(\theta)}{\partial \theta_k} \approx \frac{\mathcal{J}(\theta + \epsilon u_k) - \mathcal{J}(\theta)}{\epsilon} \]

where

\[ \nabla \mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta} [\nabla \ln \pi(a|s, \theta) Q_\pi(s, a)] \]

see REINFORCE and Actor-Critic, which are full learning algorithms that do this. See [an excellent write-up by Lilian Weng](Link to an excellent write-up by Lilian Weng) on policy-gradient algorithms.
Common Approaches

Evolution Strategies

If we satisfy:

1. Solutions able to freely interact with the environment and see what they do
2. Fitness for any solution can be evaluated

... then we can use evolution strategies (e.g. CMA-ES, Genetic Algorithm) using a non MDP-based approach without value approximation.

We assume a prior distribution over the policy parameters $\theta$ (e.g. a multivariate Gaussian) and sample the gradient accordingly

$$\nabla_\theta \mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma \epsilon) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,I)} [F(\theta + \sigma \epsilon) \epsilon]$$
Take away points

- Reinforcement learning is a huge field still in its infancy...
  - There are lots of ways to build RL agents
  - There are lots of constraints in different applications
  - Quite expensive (compute, training examples)
  - Exploitation vs exploration
- We've only dipped our feet into it!
1. Continuous form of policy reward function

\[ J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi(a|s; \theta) Q_\pi(s, a) \]

\[ \nabla J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \nabla \pi(a|s; \theta) Q_\pi(s, a) \]

\[ = \sum_{s \in S} d(s) \sum_{a \in A} \pi(a|s; \theta) \frac{\nabla \pi(a|s; \theta)}{\pi(a|s; \theta)} Q_\pi(s, a) \]

\[ = \sum_{s \in S} d(s) \sum_{a \in A} \pi(a|s; \theta) \nabla \ln \pi(a|s; \theta) Q_\pi(s, a) \]

\[ = \mathbb{E}_{\pi_\theta}[\nabla \ln \pi(a|s; \theta) Q_\pi(s, a)] \]

2. Lerp value-return for TD-learning

\[ V(S_t) \leftarrow (1 - \alpha)V(S_t) + \alpha G_t \]

\[ V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \]

\[ V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \]
$s$ state
$a$ action
$S$ set of all nonterminal states
$S^+$ set of all states, including the terminal state
$A(s)$ set of actions possible in state $s$
$R$ set of possible rewards

t discrete time step
$T$ final time step of an episode
$S_t$ state at $t$
$A_t$ action at $t$
$R_t$ reward at $t$, dependent, like $S_t$, on $A_{t-1}$ and $S_{t-1}$
$G_t$ return (cumulative discounted reward) following $t$
Appendix

Notation 2/3

\( \pi \) policy, decision-making rule

\( \pi(s) \) action taken in state \( s \) under deterministic policy \( \pi \)

\( \pi(a|s) \) probability of taking action \( a \) in state \( s \) under stochastic policy \( \pi \)

\( p(s', r|s, a) \) probability of transitioning to state \( s' \), with reward \( r \), from \( s, a \)

\( v_\pi(s) \) value of state \( s \) under policy \( \pi \) (expected return)

\( v_*(s) \) value of state \( s \) under the optimal policy

\( q_\pi(s, a) \) value of taking action \( a \) in state \( s \) under policy \( \pi \)

\( q_*(s, a) \) value of taking action \( a \) in state \( s \) under the optimal policy

\( V_t(s) \) estimate (a random variable) of \( v_\pi(s) \) or \( v_*(s) \)

\( Q_t(s, a) \) estimate (a random variable) of \( q_\pi(s, a) \) or \( q_*(s, a) \)

\( \hat{v}(s, \mathbf{w}) \) approximate value of state \( s \) given a vector of weights \( \mathbf{w} \)

\( \hat{q}(s, a, \mathbf{w}) \) approximate value of state–action pair \( s, a \) given weights \( \mathbf{w} \)

\( \mathbf{w}, \mathbf{w}_t \) vector of (possibly learned) weights underlying an approximate value function

\( \mathbf{x}(s) \) vector of features visible when in state \( s \)
### Notation 3/3

- $\delta_t$: temporal-difference error at $t$ (a random variable, even though not upper case.
- $E_t(s)$: eligibility trace for state $s$ at $t$
- $E_t(s,a)$: eligibility trace for a state–action pair
- $e_t$: eligibility trace vector at $t$

Other notations used in some fields

- $\gamma$: discount-rate parameter
- $\varepsilon$: probability of random action in $\varepsilon$-greedy policy
- $\alpha, \beta$: step-size parameters
- $\lambda$: decay-rate parameter for eligibility traces

---

**David Silver’s Slides**

[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)

**Lilian Weng’s Blog**


<table>
<thead>
<tr>
<th>State</th>
<th>s</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>a</td>
<td>u</td>
</tr>
<tr>
<td>Reward</td>
<td>r</td>
<td>-c</td>
</tr>
</tbody>
</table>