DEEP REINFORCEMENT LEARNING

Oliver Schulte
Simon Fraser University
CMPT 728
Introduction to Deep Learning
OUTLINE

• Reinforcement Learning as Function Learning
• Deep RL using Episodes
  • Monte Carlo Learning
  • Temporal Difference Learning
  • Policy Gradient
  • Actor-Critic
REINFORCEMENT LEARNING AS FUNCTION LEARNING
STATE-BASED FUNCTIONS

- RL goal is to learn functions that map states to outputs
- Tabular/grid representation: 1 row per state
- Factored representation: state = list of features
- Neural net: #of features = #input nodes

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>V(s)</th>
<th>Q(s, a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>A1</td>
<td>0.345</td>
<td>0.012</td>
</tr>
</tbody>
</table>
These 3 functions can be computed by neural networks.

$r(s, a)$ (immediate reward) values

$Q(s, a)$ values

$V^*(s)$ values

One optimal policy
WHEN TO USE FUNCTION APPROXIMATORS?

• State space is too large to handle by tabular representation
  • E.g. #possible chess positions > #atoms in the universe
• Curse of dimensionality: linearly more features → exponentially more states
• States/actions are continuous
  • E.g. location, time in sports
  • Choose location, speed in puckworld
EXAMPLES
TOY EXAMPLE: CART POLE

- **Open AI virtual environment**
- State = 4 numbers
- \((\text{pos cart } t\!-\!1, \text{angle } t\!-\!1, \text{pos cart } t, \text{angle } t)\)
- Move Left or Right
- Reward = 1 for move that doesn’t topple the cart

Figure 6.8: A cart pole
GAMES

• Board Games: input position of pieces
  • TD-Gammon for backgammon
  • Also chess, checkers

• Video Games: input (sequence of) video frames
  • Use CNN, maybe combined with LSTM
  • Starcraft
EXAMPLE: ALPHA* GAMES

• data generated by self-play
• Neural net outputs 2 quantities
  1. \( V(s) \), the win rate from a position
  2. \( P(a|s) \): vector of move probabilities
    • more promising moves should have higher probability
    • Like node ordering in tree search
• To play, performs a (Monte Carlo) tree search using the neural net output
• Watch the alphago [movie](https://www.youtube.com/watch?v=sk219z5V-FM)
ICE HOCKEY EXAMPLES

- I’ve done quite a lot of work applying RL to ice hockey
- Using millions of events from NHL games

<table>
<thead>
<tr>
<th>GID</th>
<th>PID</th>
<th>GT</th>
<th>TID</th>
<th>X</th>
<th>Y</th>
<th>MP</th>
<th>GD</th>
<th>Action</th>
<th>OC</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1365</td>
<td>126</td>
<td>14.3</td>
<td>6</td>
<td>-11.0</td>
<td>25.5</td>
<td>Even</td>
<td>0</td>
<td>Lpr</td>
<td>S</td>
<td>A</td>
</tr>
<tr>
<td>1365</td>
<td>126</td>
<td>17.5</td>
<td>6</td>
<td>-23.5</td>
<td>-36.5</td>
<td>Even</td>
<td>0</td>
<td>Carry</td>
<td>S</td>
<td>A</td>
</tr>
<tr>
<td>1365</td>
<td>270</td>
<td>17.8</td>
<td>23</td>
<td>14.5</td>
<td>35.5</td>
<td>Even</td>
<td>0</td>
<td>Block</td>
<td>S</td>
<td>A</td>
</tr>
<tr>
<td>1365</td>
<td>126</td>
<td>17.8</td>
<td>6</td>
<td>-18.5</td>
<td>-37.0</td>
<td>Even</td>
<td>0</td>
<td>Pass</td>
<td>F</td>
<td>A</td>
</tr>
<tr>
<td>1365</td>
<td>609</td>
<td>19.3</td>
<td>23</td>
<td>-28.0</td>
<td>25.5</td>
<td>Even</td>
<td>0</td>
<td>Lpr</td>
<td>S</td>
<td>H</td>
</tr>
<tr>
<td>1365</td>
<td>609</td>
<td>19.3</td>
<td>23</td>
<td>-28.0</td>
<td>25.5</td>
<td>Even</td>
<td>0</td>
<td>Pass</td>
<td>S</td>
<td>H</td>
</tr>
</tbody>
</table>

GID=GameId, PID=playerId, GT=GameTime, TID=TeamId, MP=Manpower, GD=Goal Difference, OC = Outcome, S=Succeed, F=Fail, P = Team Possess puck, H=Home, A=Away, H/A=Team who performs action, TR = Time Remain, PN = Play Number, D = Duration
PIPELINE

- Computer Vision Techniques: Video tracking
- Play-by-play Dataset
- Large-scale Machine Learning
Spatial Projection

Q-value for the action “shot” action over the rink.
Value Ticker: Temporal Projection

Game Time

Penguins Scored

Blue Jackets Scored

Game ended

Blue Jackets advantages, tried lots of shots

Penguins passed around Blue Jackets’ goal, shot
MODEL-FREE LEARNING
THE PROBLEM WITH TRANSITION PROBABILITIES

• Value iteration computes value functions, policy given transition probabilities

• In continuous/massive state spaces, working with transition probabilities is hard. Why?
  • Each transition occurs only once in data → cannot estimate probability
  • Cannot represent in matrix
  • Cannot sum over all states, actions (recall Bellman equation)
LEARNING WITHOUT TRANSITION PROBABILITIES

• "model-free" learning = no transition probabilities
  • Unfortunate term, because we use a NN model to do “model-free” RL
• Learn target function directly from episodes
  • (Many people who know Markov decision processes but not reinforcement learning don’t know about model-free learning)
• The main problem: what is the loss function?
  • No supervision signal
EPISODES

- A **transition** is a 5-item sequence $s, a, r, s', a'$
- An **episode** is a sequence of transitions $s_0, a_0, r_1, s_1, a_1, \ldots, s_{n-1}, a_{n-1}, r_n$
- At every time $t$, we denote the episode return or (discounted) sum of future rewards as $v_t$.
- Examples:
  - Sports: each game is an episode. At each time $t$ in the game, the return $v_t$ specifies whether the team won or lost.
  - Tennis: each match is broken into set episodes which are broken into point episodes
  - Degree: each course is broken into components.
    - The return $v_t$ specifies learning outcomes/grade at the end of course.
### Driving Home Example

<table>
<thead>
<tr>
<th>State</th>
<th>Elapsed Minutes</th>
<th>(v_t): future time</th>
<th>Elapsed Minutes</th>
<th>(v_t): future time</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office</td>
<td>0</td>
<td>43</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>5</td>
<td>38</td>
<td>5</td>
<td>39</td>
</tr>
<tr>
<td>exit highway</td>
<td>20</td>
<td>23</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>behind truck</td>
<td>30</td>
<td>13</td>
<td>31</td>
<td>13</td>
</tr>
<tr>
<td>home street</td>
<td>40</td>
<td>3</td>
<td>41</td>
<td>3</td>
</tr>
<tr>
<td>arrive home</td>
<td>43</td>
<td>0</td>
<td>44</td>
<td>0</td>
</tr>
</tbody>
</table>

- Could have different states/actions in different episodes
  - E.g. some days no rain

Deep Reinforcement Learning
MONTE CARLO LEARNING

Regression approach
REGRESSION APPROACH

• Start with a dataset of episodes
• For every observed state $s_t$, make the return $v_t$ the target.
• Train a model to predict (the expected value of) $v_t$ given $s_t$ as input.
**REGRESSION APPROACH: EXAMPLE**

<table>
<thead>
<tr>
<th>State = $X$</th>
<th>$v_t = Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office</td>
<td>43</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>38</td>
</tr>
<tr>
<td>exit highway</td>
<td>23</td>
</tr>
<tr>
<td>behind truck</td>
<td>13</td>
</tr>
<tr>
<td>home street</td>
<td>3</td>
</tr>
<tr>
<td>arrive home</td>
<td>0</td>
</tr>
<tr>
<td>leaving office</td>
<td>44</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>39</td>
</tr>
<tr>
<td>exit highway</td>
<td>22</td>
</tr>
<tr>
<td>behind truck</td>
<td>13</td>
</tr>
<tr>
<td>home street</td>
<td>3</td>
</tr>
<tr>
<td>arrive home</td>
<td>0</td>
</tr>
</tbody>
</table>

- Regression: Predict $Y$ given $X$
- People who know machine learning but not reinforcement learning use this approach
- Sports examples:
  - How to serve in volleyball
  - [Valuing Actions by Estimating Probabilities](#) in soccer
- Main Problem: regression methods assume i.i.d data
  - But subsequent states have highly correlated return values
  - Slow learning
TEMPORAL DIFFERENCE LEARNING

Learning a Value Function
TD LEARNING

• A key idea in CS RL, used in almost all RL AI systems
• Key intuition: compare 2 estimates
  1. Current return estimate $V(s_t)$
  2. Return estimate from look ahead 1 time step: $r_{t+1} + \gamma V(s_{t+1})$
  3. TD-error for NN gradient = $[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]^2$
• Can be generalized to look-ahead multiple time steps
<table>
<thead>
<tr>
<th>State</th>
<th>Elapsed Minutes total</th>
<th>Current Elapsed Minutes</th>
<th>Predicted Time to Go</th>
<th>TD-target</th>
<th>TD-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>5+35 = 40</td>
<td>(40-30)^2=100</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>5</td>
<td>5</td>
<td>35</td>
<td>15+15 = 30</td>
<td>(30-35)^2 = 25</td>
</tr>
<tr>
<td>exit highway</td>
<td>20</td>
<td>15</td>
<td>15</td>
<td>10+10 = 20</td>
<td>(20-15)^2=25</td>
</tr>
<tr>
<td>behind truck</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>10+3 = 13</td>
<td>(13-10)^2=9</td>
</tr>
<tr>
<td>home street</td>
<td>40</td>
<td>10</td>
<td>3</td>
<td>3+0 = 3</td>
<td>(3-3)^2=0</td>
</tr>
<tr>
<td>arrive home</td>
<td>43</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### EXAMPLE

<table>
<thead>
<tr>
<th>State</th>
<th>Elapsed Minutes</th>
<th>Predicted Time to Go</th>
<th>Predicted Total Time</th>
<th>TD-target</th>
<th>TD-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>40</td>
<td>$(40-30)^2=100$</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>5</td>
<td>35</td>
<td>40</td>
<td>35</td>
<td>$(35-40)^2 = 25$</td>
</tr>
<tr>
<td>exit highway</td>
<td>20</td>
<td>15</td>
<td>35</td>
<td>40</td>
<td>$(40-35)^2=25$</td>
</tr>
<tr>
<td>behind truck</td>
<td>30</td>
<td>10</td>
<td>40</td>
<td>43</td>
<td>$(43-40)^2=9$</td>
</tr>
<tr>
<td>home street</td>
<td>40</td>
<td>3</td>
<td>43</td>
<td>43</td>
<td>$(43-43)^2=0$</td>
</tr>
<tr>
<td>arrive home</td>
<td>43</td>
<td>0</td>
<td>43</td>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>
TD VS. MC (REGRESSION)

Changes recommended by Monte Carlo methods ($\alpha=1$)

Changes recommended by TD methods ($\alpha=1$)

TD Pros

- Exploits temporal dependencies rather than ignoring them
- Can start learning before end of episode
- Deep versions converge faster than MC
- Technically: TD has lower variance, higher bias
POLICY LEARNING
A policy maps a state to a distribution over a finite set of actions

- Like multi-class learning
- Can be represented in a NN
- What is the objective function?

The diagram shows a deep learning architecture for REINFORCE. The important thing to notice is that the NN is used in two different ways. First, looking at the left-hand side, we give the NN a single state which, as mentioned earlier, is a four-tuple of reals indicating the position and velocity of the cart and the pole-head. In this mode we get out probabilities for taking the two possible actions, as indicated in the middle-right of the figure. When in this mode we do not provide the placeholder for the rewards or actions with values since (a) we don't know them, and (b) we don't need them since we are not computing the loss at this point. After we have made all the moves for an entire game, we use the NN in the other mode. This time we give it the sequence of actions and the rewards, and this time we ask it to compute the loss and perform back propagation. In training mode we are, in a sense, computing actions in two different ways. First, we give the NN the states we go through, and for each state, the policy computation layers compute the action probabilities. Second, we directly feed in the actions taken as a placeholder. This is because when deciding on actions in game-playing...
EXAMPLE: TRAINING ROBOT TO WALK

- Goal: learn a fast AIBO walk
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by policy gradient
- Performance metric = field traversal time
FREQUENCY DECOMPOSITION OF VALUE FUNCTION

- Consider the following frequency expression for the value of a policy
  \[ V_\pi(s_0) = \sum_s P_\pi(s|s_0) \times \sum_a Q_\pi(s,a) \times \pi(a|s) \]
- \( P_\pi(s|s_0) \) is the stationary (limiting) frequency of reaching \( s \) starting from \( s_0 \)
- We have seen an efficient dynamic programming algorithm for computing this
ANALYZING POLICIES

- The frequency decomposition is often a nice way to explain the performance of an agent

\[ V^\pi(s_0) = \sum_s P^\pi(s|s_0) \times \sum_a Q^\pi(s,a) \times \pi(a|s) \]

What will the agent do in state \(s\)?

- How often does the agent reach state \(s\)?
- How well does the agent do in state \(s\)?

Hockey Example

- How often does a team achieve a powerplay state \((s)\)?
- How successful are they with powerplay \((V^\pi(s))\)?
OPTIMIZING POLICIES

\[ V^\pi(s_0) = \sum_s P^\pi(s|s_0) \times \sum_a Q^\pi(s,a) \times \pi(a|s) \]

- Estimate observed #visits to \( s_0 \)
- Estimate from data
- Optimize
POLICY OPTIMIZATION OPTIONS

• Monte Carlo: \( \text{argmax}_\pi \sum_t \pi(a_t|s_t) \times v_t(s_t,a_t) \) where \( v_t(s_t,a_t) \) is the observed episode return as in Monte Carlo learning

• Actor-Critic: \( \text{argmax}_\pi \sum_t \pi(a_t|s_t) \times Q'(s_t,a_t) \) where \( Q' \) is a value function estimated by the critic
  • Typically using TD-learning

• Advantage Actor-Critic
  • replace \( Q'(s_t,a_t) \) by \( Q'(s_t,a_t) - V'(s_t) = \text{“advantage”}/\text{impact} \)
  • Advantage values have lower variance than \( Q \) values \( \rightarrow \) easier to learn
  • Current state of the art
Assume the policy is parametrized as $\pi_\theta(a_t|s_t)$.

The gradient can be expressed by the score function $\Delta_\theta \log(\pi_\theta(a_t|s_t))$:

$$\Delta_\theta \pi_\theta(a_t|s_t) = \pi_\theta(a_t|s_t) \times \Delta_\theta \log(\pi_\theta(a_t|s_t))$$

Demo
EXAMPLE: REINFORCE ALGORITHM

1. Initialize $\theta$

2. For each episode $<s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T>$ do
   for $t=1$ to $T-1$ do
     $\theta := \theta + \alpha \Delta \theta \log(\pi_\theta(a_t|s_t)) \, v_t(s_t, a_t)$
   end for
end for

3. Return $\theta$  
   Learning rate  
   Estimated Return
DATA GATHERING ISSUES
A fundamental difference between RL and passive learning is that the agent’s actions influence their observations.

- No fixed dataset to analyze
- Actions influence not only the rewards but also the quality of the information the agent receives
- Hockey example: maybe we need to place risky bets to get information about the payoffs?
EXPLORATION VS. EXPLOITATION

- An agent needs to do both
  - Select actions that seem optimal to keep high rewards
    - “exploit” its current knowledge
  - Select new actions to gather enough data to estimate a value function
    - “explore” the state space
BASIC EXPLORATION APPROACHES

- A simple but often effective approach is $\epsilon$-greedy
  - With probability $\epsilon$, select a random action (e.g. $\epsilon = 10\%$ of the time)
  - With probability $1-\epsilon$, select an action that is optimal according to the current value function
- $\epsilon$-decreasing: decrease $\epsilon$ with every time step
  - Like a learning rate
- Policy-driven: sample actions from probabilistic policy
  - Works best with policy gradient methods
- State-of-the-Art: Upper Confidence Bound (UCB)
  - Estimate uncertainty in value functions at a state
  - Visit states with more uncertainty more often
• Recall that temporally successive states have highly correlated values
  ➢ Successive updates change value functions only little
• Possible solution:
  1. Store transitions $s_{t-1}, a_{t-1}, r_t, s_t, a_t$
  2. Sample randomly from past transitions to update
• But is throwing away temporal information really a good thing?
SUMMARY OF APPROACHES

- Value-based
  - Learn Value Function
  - Implicit policy (e.g. ε-greedy)
- Policy-based
  - No Value Function
  - Learnt policy
- Actor-Critic
  - Learn value function
  - Learnt policy
SUMMARY RL

- Reinforcement learning aims to learn various key functions
  - Value function, policy
  - Can be naturally implemented as neural networks
  - In large/infinite state spaces, need model-free learning
- Learning techniques for value functions
  - Monte-Carlo: reduce to regression
  - Temporal difference: make different value estimates at different times consistent with each other