DEEP REINFORCEMENT LEARNING

RESHMI GHOSH,

DATA AND APPLIED SCIENTIST II, MICROSOFT

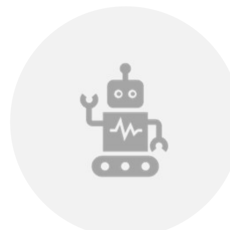
ABOUT ME

- Currently a Data & Applied Scientist, Microsoft
- Ph.D. & M.S. from Carnegie Mellon University
- Teaching Assistant for 11-785 in Fall 2020 and Spring 2021
- Thesis on stochastic modeling & novel Deep Learning based sequential analysis to make energy systems more robust under climate change

Bhiksha's fan ☺ and believer of meme threads on Piazza

HOW IS REINFORCEMENT LEARNING DIFFERENT FROM ML/DL?

Statistics - Understand data distribution, the dispersion of data points, methods to tell how two groups of data are different



Machine Learning - Based on existing data representation – you teach a ‘machine’ to recognize patterns and predict it

Deep Learning - A fancier version of machine learning where your machine recognizes decision boundaries which are non-linear in nature – a function approximator

***Reinforcement Learning* - Teach a ‘machine’ how to decide (ACTION) based on a ‘prior’ understanding of the environment (STATE) and possible outcomes (REWARDS) of the decision(subset of ML/DL algorithms)**

REINFORCEMENT LEARNING FRAMEWORK

ACTION

What does an agent decide to do?

REWARDS

Is the decision right or wrong?

ENVIRONMENT

In what conditions is the agent operating?

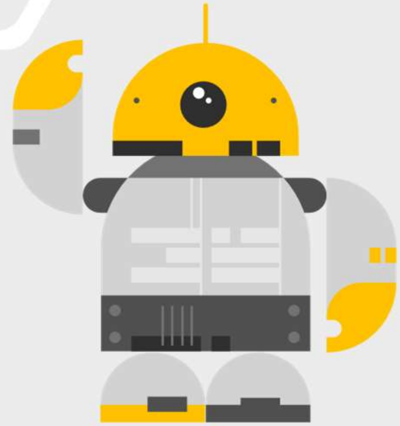
STATES

What are possible 'environments' the agent can transition into after making a decision?

Reinforcement Learning Examples

- A game of chess
  - States:
  - Actions: Which move to choose?
  - Reward: What do you gain or lose?

- Robots completing a certain task
  - States: Observation from sensors
  - Actions: What is the next move?
  - Reward:
    - Appreciation for the correct move towards completion of task
    - Penalty for unexpected moves, accidents



Formulation

- Markov Decision Process (MDP) vs. Q-Learning
- States S
- Actions A (not all actions possible in every state**)
- Reward function $R\colon\ A\ X\ S\ \rightarrow \mathcal{R}$
- Transition function
- Sequence of interactions between agent and environment
  - s_1, a_1, R_1, s_2, a_2, R_2, ...

WHAT IS OPTIMAL POLICY?

To make the agent behave optimally – it needs to learn how to choose the best action for each state, i.e., optimal policy

Type equation here.

Mapping

A mapping from past experiences to action that is also tractable (cannot be a set of if-else statements)

Infinite Horizon: include a discounting factor

Discount factor: γ

Is finite horizon used?

Average reward definitions of optimality are rare

Maximize the ‘expected’ rewards

$$\boldsymbol{E(\sum_t^\infty \gamma^t R_t)}$$

AI PLANNING: EXPLORATION Vs. EXPLOITATION

- How does agent learn optimal policy?
  - Needs to evaluate every possible action for all (infinite) states.
  - What if the agent chooses an action that it thinks is best all the time? – unexplored parts of state space ignored.
  - What if the agent chooses to explore extensively? How to leverage (exploit) what the agent has learned?
  - Delayed consequences

Optimal Policy

Value of a state: expected infinite cumulative reward that has been discounted after agent applies the 'optimal policy'

$$V(s) \ = \ \max \boldsymbol{E}(\sum_{\boldsymbol{t}}^{\infty} \boldsymbol{\gamma}^{\boldsymbol{t}} \boldsymbol{R}_{\boldsymbol{t}})$$

Using the value function, define the optimal policy

$$\pi^* \ (s) = \arg\max(R(s,a) + \ \gamma \sum_{s' \epsilon \ S} T(s,a,s')V^*(S'))$$

Q-learning over MDP

No transition function – approximate value of each state based on the actions taken and rewards/penalty received

Q*(s, a) -> expected discounted reward of choosing action a in state s, and choosing the next actions optimally

$$Q^*(s,a) = \quad R(s,a) + \gamma \sum_{s' \epsilon S} T(s,a,s') \max Q^* (s',a'))$$

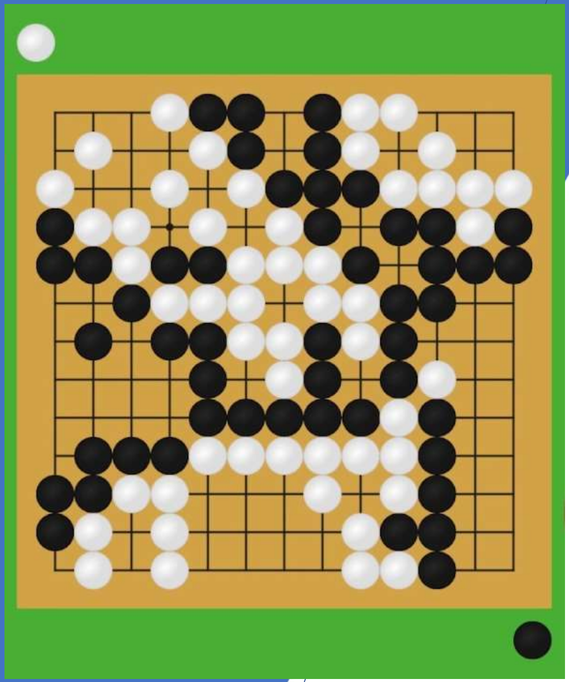
$$\pi^*(s) = argmax\ Q^*(s,a)$$

Observe transition (s, a, r, s')

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max Q(s',a'))$$

**Guaranteed to converge

COMPLEX SCENARIO - GO GAME



19X19 DIFFERENT POSITIONS

3 DIFFERENT STRATEGIES – BLACK STONE, WHITE STONE, NO STONE

POSSIBLE NUMBER OF STATES? 3^{19X19}

Impossible to learn the transition matrix – instead approximate it

WHAT IS THE BEST WAY TO APPROXIMATE A FUNCTION?

Hence, Deep Reinforcement learning

State and actions are not always discrete + difficult to store all possible states and action values

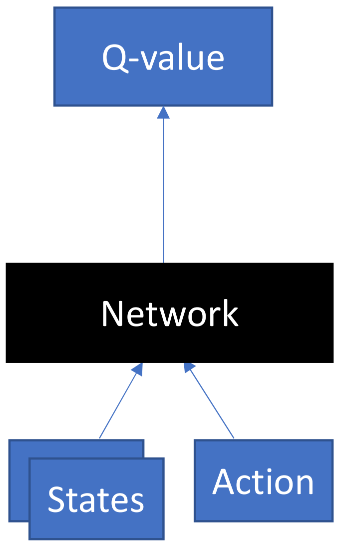
**Convergence is not guaranteed for models with function approximation

Deep Q-Learning

Use DNN to approximate Q functions

Use observed trajectories of state-action-rewards

'Human level control through Deep Reinforcement Learning', 2015 – by DeepMind



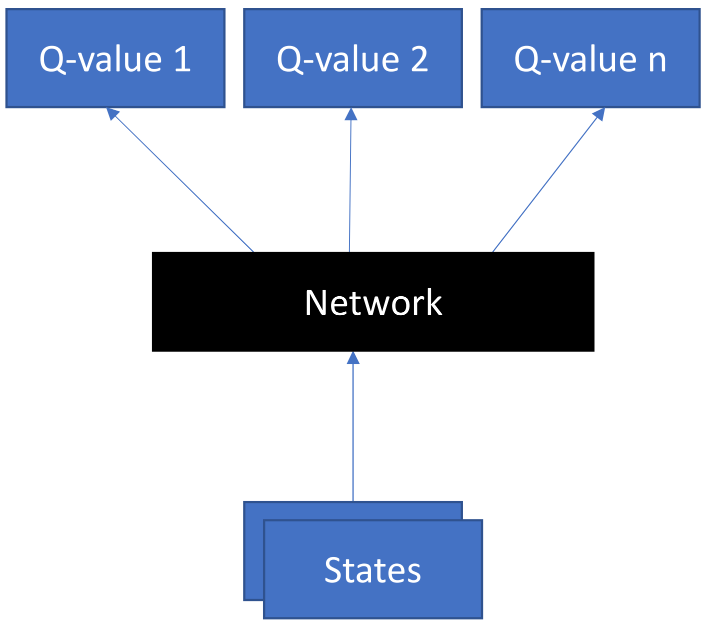
Feed the network with state-action pair to output a corresponding Q-value

Deep Q-Learning

Use DNN to approximate Q functions

Use observed trajectories of state-action-rewards

'Human level control through Deep Reinforcement Learning', 2015 – by DeepMind



Compute Q-values for all actions in one go

Deep Q-Learning formulation

Formulate the value function by Deep Q-network with weights w

Objective function – MSE in Q values

Algorithm

Transition -> (s, a, r, $s^{'}$)

1. Feedforward pass for current state s to get predicted Q-values for all actions
2. Feedforward pass for next state $s^{'}$ and calculate argmax across all network outputs argmax Q($s^{'}$, $a^{'}$)
3. Assign Q-value target for action to $r + \gamma \max Q(s', a')$
4. Update weights during backprop

Pitfalls

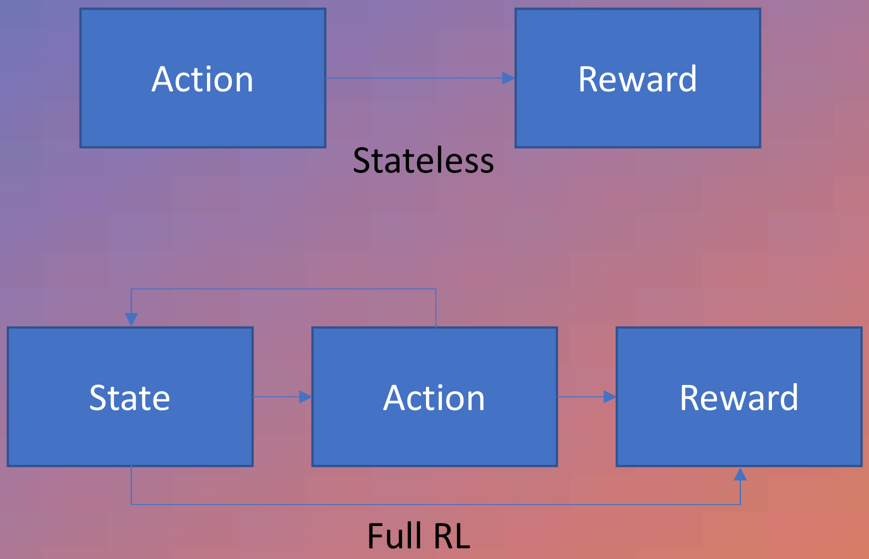
Pitfall	Solution
Policy changes frequently with small changes to Q-values, may oscillate	Compute Q-learning targets w.r.t old, fixed parameters w
Scale of rewards and Q-values – unknown, thus gradients can be large and unstable	Clip rewards or normalize network adaptively

Problems with stability

Q-learning diverges in presence of:

- Off-policy learning
  - Learning from data generated by a policy about a different policy – better value approximation for the policy actually being implemented
- Bootstrapping
  - Learning value estimates from other value estimates

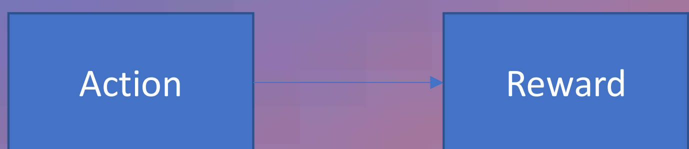
Multi-arm Bandits



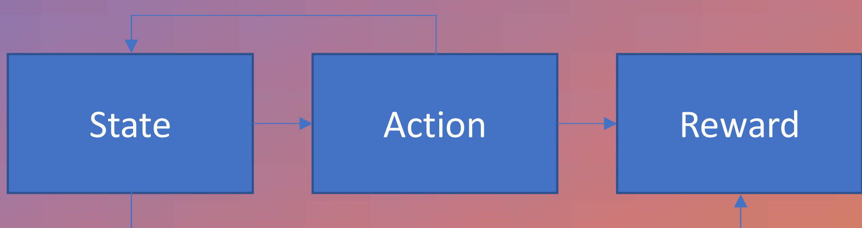
- Slot machine – one armed bandit
- A multi-arm bandit is a row of non-identical slot machines
- limited set of resources must be allocated to maximize reward

Infinite-Arm Bandits -Do I keep training this hyperparameter configuration, or do I check others to see if they give better performance?

Multi-arm Bandits



Stateless



Full RL

A **multi-armed bandit** (also known as an N-**armed bandit**) is defined by a set of *random variables* $X_{i,k}$ where:

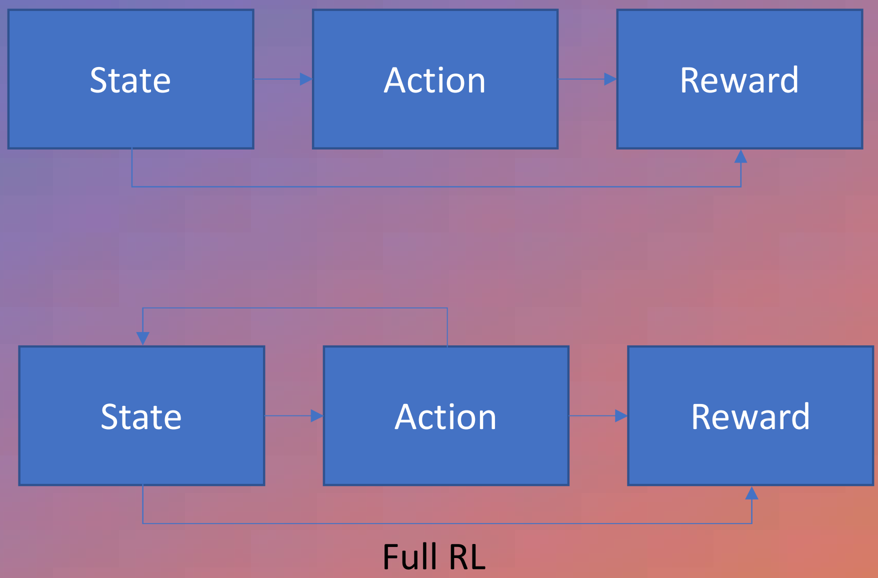
- $1 \leq i \leq N$, such that i is the *arm* of the bandit; and
- k the index of the *play* of arm i;

Successive plays $X_{i,1}, X_{j,2}, X_{k,3} \ldots$ are assumed to be independently distributed, but we do not know the probability distributions of the random variables.

Then, the Q-value for an action a can be estimated using the following formula:

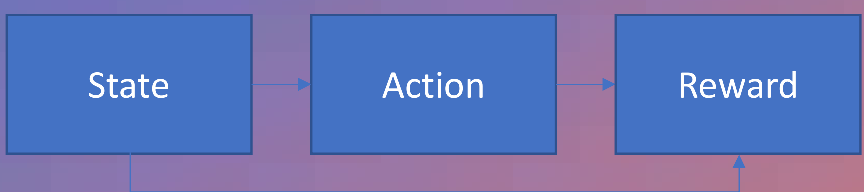
$$Q(a) = \frac{1}{N(a)} \sum_{i=1}^{t} \mathbb{I}_i(a) r_i$$

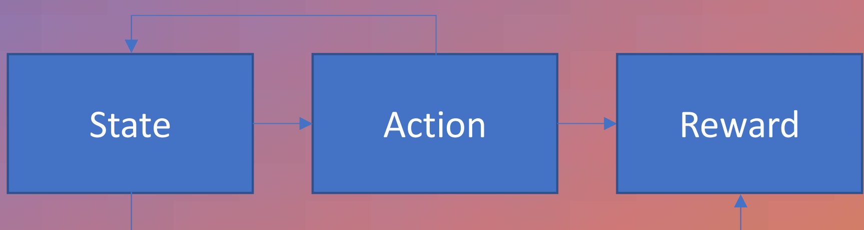
Contextual Bandits



- Agent has access to additional information at beginning of each round
- Specializing the action to context – better rewards

Contextual Bandits





Full RL

Below are the steps in the learning process:

1. Observe context, x_t, at time t
2. Select action, $a_t \in \{1, . . . , K\}$, with the help from a policy/policies, π_t, where $\pi_t(x_t) = a$
   a. Estimate the reward of each action conditional on the context
   b. Generate a probability mass function (pmf) over the actions based on their estimated rewards
   c. Randomly choose the action according to probability, p_a
3. Observe the reward, $r_t(a_t)$ (alternatively, can be denoted as loss $l_t(a_t)= -r_t(a_t)$)
4. Update the learner's policy/policies based on the quadruple (x_t, a_t, p_{a_t}, r_t) to obtain π_{t+1}
5. Repeat steps 1-4

Contextual Bandits

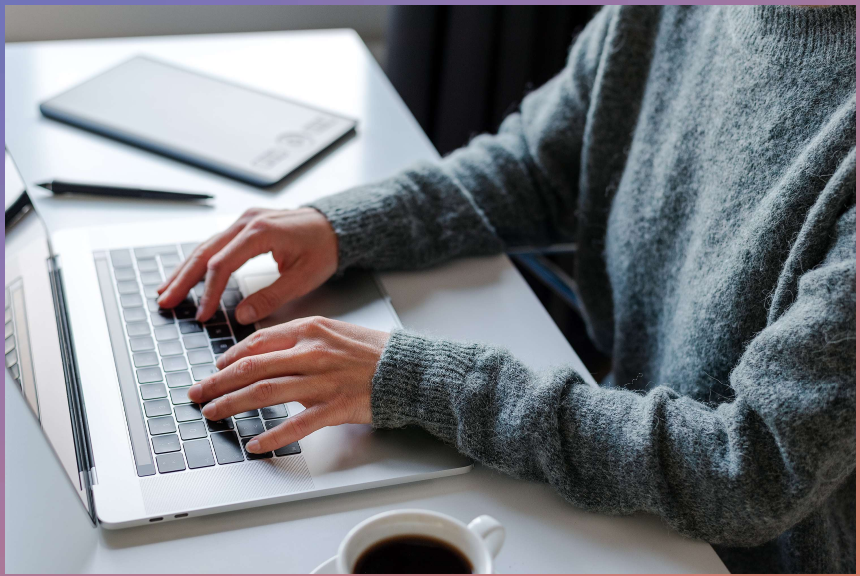
Case study: Recommending news articles when you open a web page

State: User interests, news article specifics

Action: category of news article – finance, health, sports etc.

Reward: click/no click

Contextual Bandits



Case study: Which ads to show to a user?

State: User interaction with website

Context: User demographics + preferences

Action: Ad types

Reward: Click on ad/No Click on ad

Contextual Bandits – Product use cases - Why not a conventional recommendation system?

Goal: Improve customer experience and engagement with personalized ranking

Challenges:

1. Non-stationary content pool
2. Change in user preferences over time – how to adapt?

Solution:

- Optimize the CTR with contextual bandits

QUESTIONS