Multi-task Learning with Reinforcement Learning Methods

Ryan Hilton, Kevin Xue, Nate Whybra, Bowen Jin, Jiacan Yu

August 14, 2022

UNIVERSITY of ROCHESTER

Table of Contents

Introduction to RL

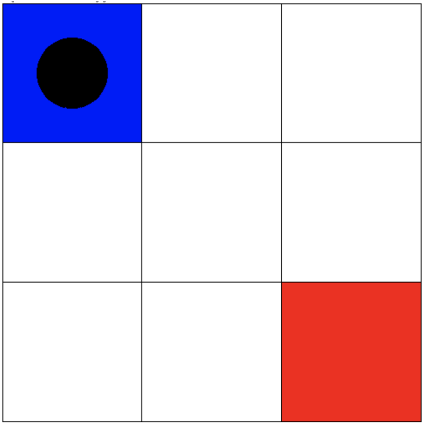
Definition

- Reinforcement learning (RL) is machine learning area alongside supervised learning and unsupervised learning.
- "It is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment." [1]

GridWorld

Task 1: Targeting Game

Goal: reach the terminal state with the least number of steps



States:

- start_state: blue
- terminal_state: red
- agent_position: black circle

Actions:

move up, left, down, and right

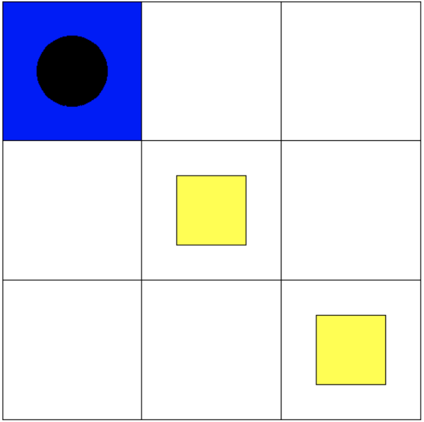
Reward policy:

- if move out of the board, get -1000
- if move to the terminal_state, get 0
- for any other legal moves, get -1

Terminate when the agent reaches the terminal_state.

Task 2: Collection Game

Goal: collect all the prizes with the least number of steps



States:

- start_state: blue
- remaining_prize_state: yellow square
- agent_position: black circle

Actions:

move up, left, down, and right

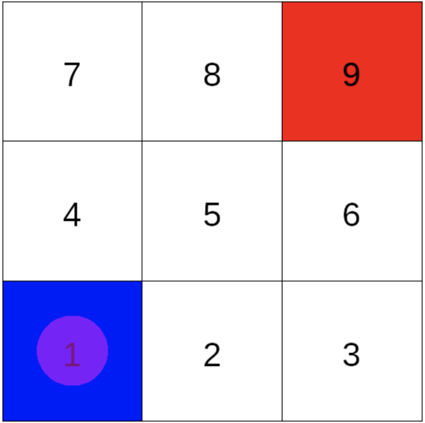
Reward policy:

- if move out of the board, get -1000
- if move to a remaining_prize_state, get 0
- for any other legal moves, get -1

Terminate when the agent collects all the prizes.

Task 3: FindMax Game

Goal: reach the max value with the least number of steps



States:

- start_state: blue
- max_state: red
- agent_position: purple circle

Actions:

move up, left, down, and right

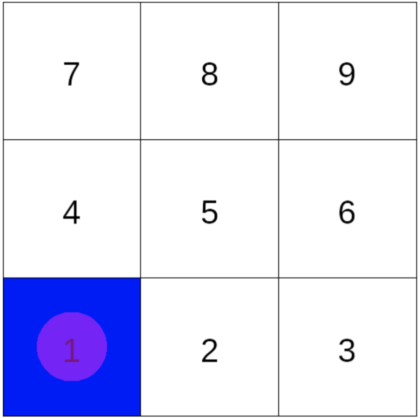
Reward policy:

- if move out of the board, get -1000
- else get the reward on board

Terminate when the agent reaches the max_state.

Task 4: MaxPath Game

Goal: maximize the sum of rewards along a n-step path



States:

- start_state: blue
- agent_position: purple circle

Actions:

move up, left, down, and right

Reward policy:

- if move out of the board, get -1000
- else get the reward on board

Terminate after the agent takes n moves.

Tabular Solution Methods

Markov Decision Process

- Markov Decision Processes (MDPs) are a classical formalization of sequential decision-making.
- Agent + Environment interact at each time step (Example Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, ...$)
- The use of a reward signal to formalize a goal is one of the most distinctive features of RL.
- Discount Rate and Returns ($G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$) : $\gamma \in [0, 1]$
- Policies ($\pi(a|s)$ and $b(a|s)$) and Value Functions. ($V_\pi(s)$ and $Q_\pi(s, a)$)
  - How good is it to perform a given action in a given state?

Monte Carlo Methods

- How do we estimate value functions and derive policies from them? Monte Carlo.
- Monte Carlo methods sample episode trajectories from agent interaction with the environment.
- Maintain average returns for each state and average and these should converge to their true values as number of samples approaches infinity.
- In Monte Carlo methods, we only update our policies and value functions after completion of each episode.
- Maintain exploration vs exploitation
  - On-Policy (one policy: $\pi(a|s)$) vs Off-policy methods (two policies: $\pi(a|s)$ and $b(a|s)$)

Temporal Difference Learning Methods

- What if we don't want to wait an entire episode before updating our policies and value functions? Temporal Difference (TD).
- one-step TD methods only need to wait till the next time step to update their models.
- one-step TD learning methods:
  - Sarsa:

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

  - Q-Learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (2)$$

  - Expected Sarsa:

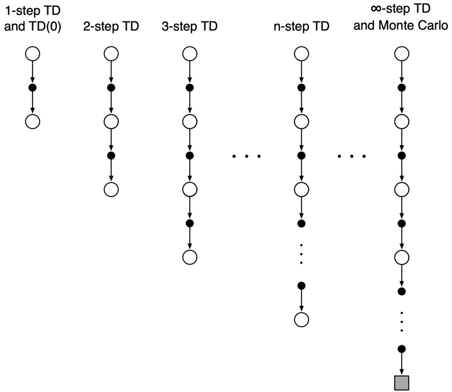
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (3)$$

n-step Bootstrapping Methods

- Monte Carlo and Temporal Difference methods are the extreme cases of policy and value function updating.
- The intermediate case is n-step TD, which allows you to decide exactly how farsighted and nearsighted.
- n-step Update Rule

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_{t:t+n} - Q(S_t, A_t)] \tag{4}$$

n-step Bootstrapping Visualization



Planning Methods

- So far we've discussed model-free methods which parameterize an environment based on real experience. We now consider model-based methods which relies on planning (real and simulated experience).
- Simulated experience is obtained from a model of the environment, and the model of the environment is obtained from the real experience.

Approximate Solution Methods

Neural Networks

- When it's too difficult to represent every state-action pair in a table, we consider approximate solutions that parameterize the tabular solution.
- Neural Networks are very flexible function approximators.
- We can approximate our value functions or the policy.
- We can only guarantee local optimum solutions for non-linear function approximators.

Policy Gradient Methods

- What if we want a parameterized policy that represents the target policy and doesn't consult a value function.
- Takes in state input and outputs action (akin to a multi-class classification NN.

Multi-Task Learning

Definition

- Crawshaw defines Multi-task learning as “subfield of machine learning in which multiple tasks are simultaneously learned by a shared model” in which humanity’s behavior is replicated by applying past situations to handle future situations. [2]
- Ex. A baby who learns how to walk begins to intuitively understand physics, lending a positive influence on its ability to balance and eventually ride a bike.
- In the more specific case in our scenario, MTL is defined as a model in which data points are shared between tasks.

Popular Methods for MTL

- Shared trunk - A singular network model in which tasks share similar features and the output of one task becomes the input of another task, producing a cascading effect in which there is an exchange of information between tasks as a result of shared parameters.
- Cross-stitching - A model with multiple networks in which "the input to each layer is a linear combination of the outputs of the previous layer from every task network" and the respective weights of each linear combination is learned from a loss or reward function. [2].

Visual Representation

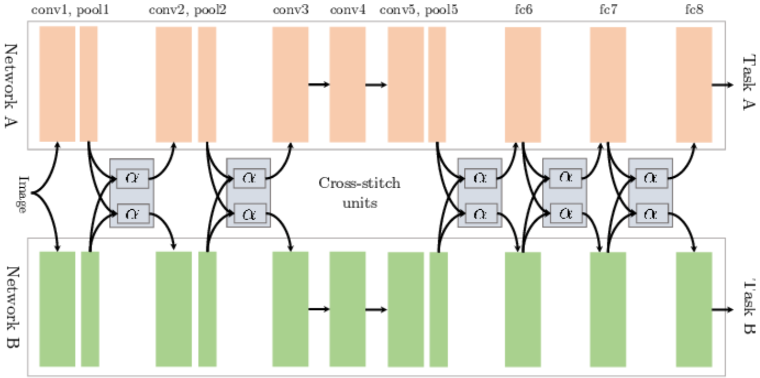


Figure: An image of a Cross-stitched network

Motivation of MTL

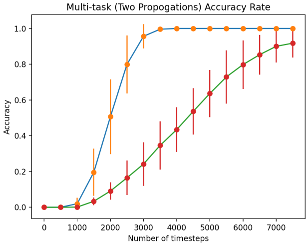
- Our goal for this project is explore machine learning in a larger context with applications to practical problems in which multiple tasks must be solved. This idea of reaching the optimal joint solution, or the Pareto optimal solution has applications in other fields, such as economics.
- In addition, making ML models more versatile and generalized can also save computational power and more finely tune the models that are currently be developed by grouping tasks with commonalities as is present in human-like learning.
- We hope to continue this exploration and find more scenarios in which MTL will improve results.

Multi-task Learning with or and xor

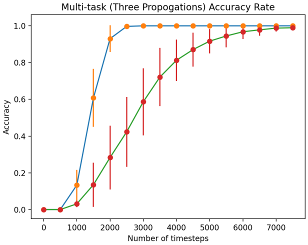
- Given 2 n bit sequences of 0s or 1s, how can we classify the output to produce the right vector when applying the "or" or the "xor" function?
- For "or," if we are given the vectors [0, 1, 1] and [0, 0, 1], we get: [0, 1, 1]
- for "xor," we get [0, 1, 0]

But how can we represent this rule within a multi-task model that can take in vectors and correctly predict the resulting vector?

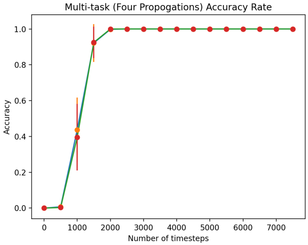
Multi-task Learning with or and xor



$$\mathcal{L}^t = \mathcal{L}^t_{or} \text{ or } \mathcal{L}^t_{xor}$$



$$\mathcal{L}^t = c_{or}\mathcal{L}^t_{or} + c_{xor}\mathcal{L}^t_{xor}$$



$$\mathcal{L}^{t1} = \mathcal{L}^t_{or},\ \mathcal{L}^{t2} = \mathcal{L}^t_{xor}$$

Conclusions

- In this case of learning the or and xor operator, multi-task learning benefits both tasks, reducing the resources to reach 100 percent accuracy while we increase the number of propagations.
- This already shows that in certain cases where the problems are similar and need similar steps to solve or to represent the function, multi-tasking is computationally efficient.
- In this case, we also find that the accuracy rate trends of both tasks converge when we increase the number of propagations and propagate both task inputs forward and loss backwards.

Multi-Task Deep Reinforcement Learning

Next Steps

- Our end goal at the moment is to investigate the idea of building a large multi task neural network and train it on multiple reinforcement learning tasks. Then identify substructures of this large network that achieve the same or, very close to the same, efficacy as neural networks built for each task individually.
- We currently are looking at two different problems related to investigate our goal.
- From the reinforcement learning side, we are investigating the problem where we train two agents on two instances of a game and find traces of similar information shared between the two trained models.
- From another side, we are trying to develop theory for updating the masks using information in back-propagation.

References I

- L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
- M. Crawshaw, “Multi-task learning with deep neural networks: A survey,” 2020.

Acknowledgment

This work was supported by the National Science Foundation under grant US NSF HDR TRIPODS 1934962.