Advanced Artificial Intelligence

Lab 13

Outline

- Discuss differences between model-based and model-free RL
- Use a concrete problem to compare Monte Carlo method with temporal difference learning
- Exercise

Model-based & Model-free

- Whether the agent **learns a model of the environment**. If so, model-based, if not, model-free.

- Here model of the environment means functions **predicts state transitions** and **rewards**, so allow agent to plan by thinking ahead

Go back to **model-based** algorithm introduced by lecture: estimate transition probability **P** and reward **R** by experience.

$$\hat{\mathcal{P}}(s,a,s') = \frac{N_{s,a,s'}}{N_{s,a}}$$

$$\hat{\mathcal{R}}(s,a,s') = \frac{R_{s,a,s'}}{N_{s,a,s'}}$$

Predict function for **P** and **R**, obviously model-based

Go back to **model-free** algorithm introduced by lecture: value function **V** or action-value function **Q** (Monte-Carlo and Temporal Difference method)

$$\hat{v}^{\pi}(S_t) \leftarrow \hat{v}^{\pi}(S_t) + \alpha\left(R_{t+1} + \gamma \hat{v}^{\pi}(S_{t+1}) - \hat{v}^{\pi}(S_t)\right)$$

Here use immediate reward **R**, but **not** maintain a predict function

Monte Carlo & Temporal Difference

Both are **model-free**, like policy iteration(PI), but use **sampling** to estimate value function or action-value function

Monte Carlo (MC)

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^k R_{t+k+1} + \cdots$$ (need sample a whole trajectory)

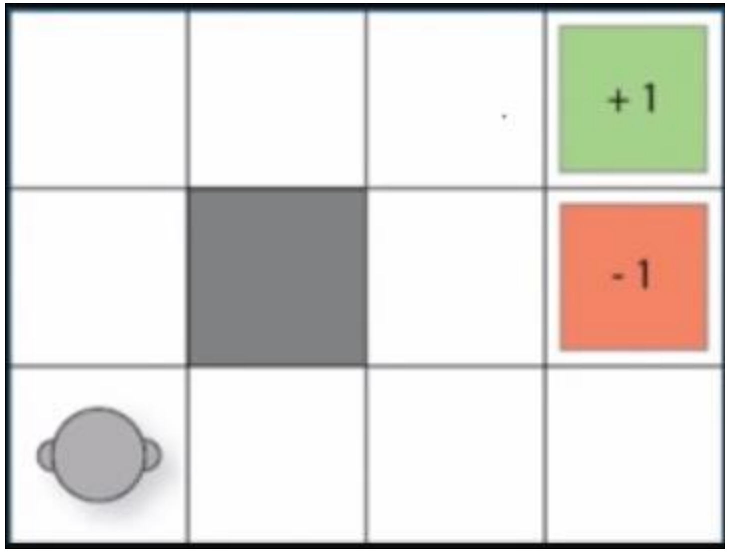
$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \alpha(G_t - \hat{v}^\pi(S_t))$$

Temporal Difference (TD)

$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \alpha\left(R_{t+1} + \gamma \hat{v}^\pi(S_{t+1}) - \hat{v}^\pi(S_t)\right)$$ (use current estimation to estimate, so biased)

Monte Carlo & Temporal Difference

GridWord:



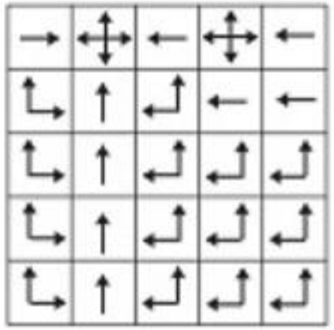
- Agent start from a place (left corner)
- Some places cannot go (grey)
- Goal place have positive reward (green)
- Some places have negative reward (red)

- **State**: agent position (a scalar, index of block)
- **Action**: move direction (UP, DOWN, LEFT, RIGHT)
- **Environment dynamics**: deterministic, leading to the same new state given each state and action
- **Reward**: 1 when goal state, -1 for some states, 0 else

With access of model of environment, we can easily construct **value function and policy graph** by PI/VI:

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

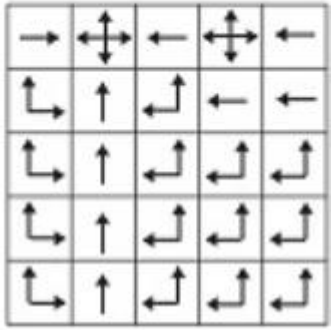
A visualization demo is here.

Monte Carlo & Temporal Difference

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*

Easy!



π_*

But how to get v^*without model of environment (model-free)?

- Monte Carlo(MC)
- Temporal Difference(TD)

Do sampling to estimate

- Estimate based on sampled trajectory
- Estimate based on sampled immediate reward and current estimation

Monte Carlo & Temporal Difference

For example, at the beginning (simple case):

		G
S		

S: start place
G: goal place

0.	0.	1.
0.		-1.
0.	0.	0.

Reward function

0.	0.	0.
0.		0.
0.	0.	0.

Value function

Aim to something like:

0.6	0.8	1.
0.4		-0.5
0.3	0.2	0.1

Value function

$\Rightarrow$

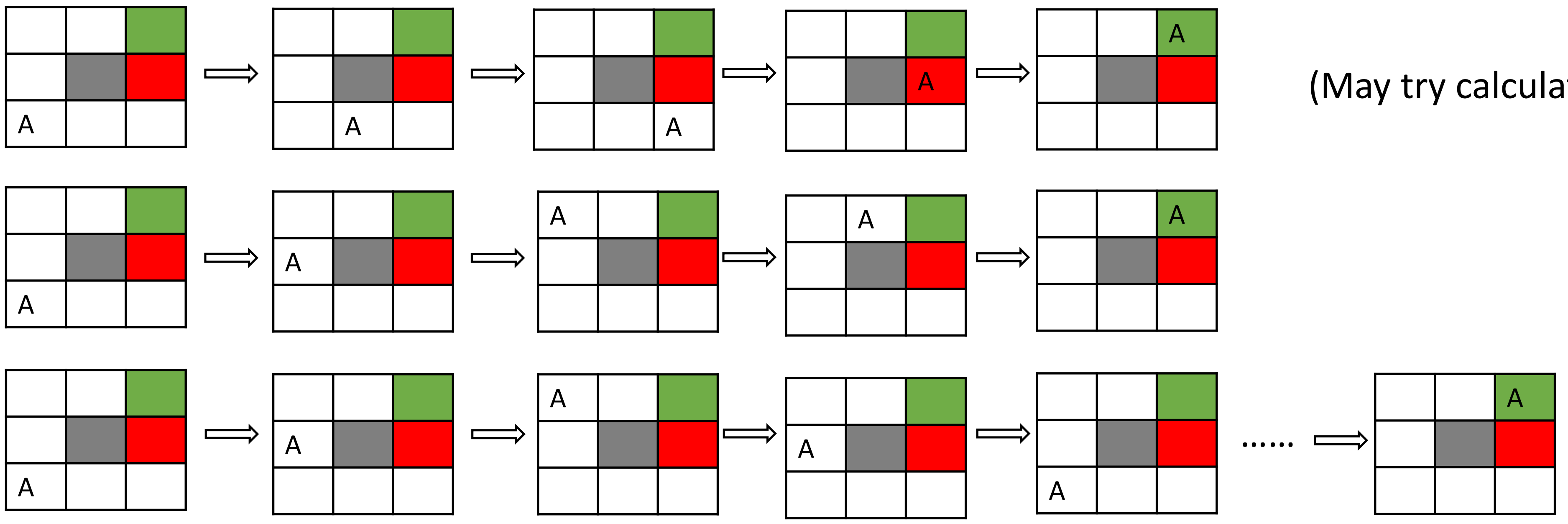
R	R	
U		U
U	L	L

Policy

R: right
U: up
L: left
D: down

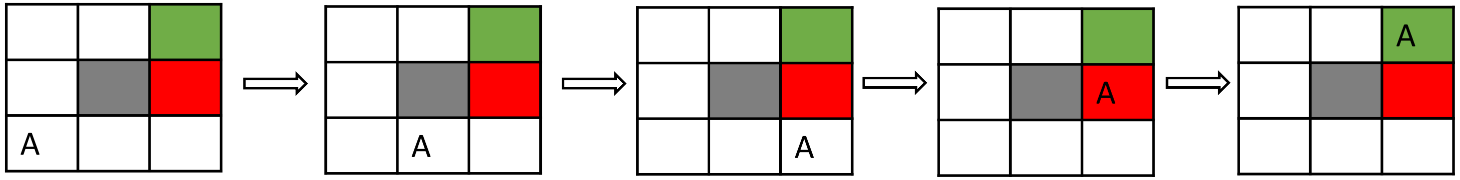
Monte Carlo & Temporal Difference

Possible sampled trajectories (A: agent):



(May try calculate this)

Monte Carlo & Temporal Difference



$R_1 = 0$	$R_2 = 0$	$R_3 = 0$	$R_4 = -1$	$R_5 = 1$
$N_1 = 1$	$N_2 = 1$	$N_3 = 1$	$N_4 = 1$	$N_5 = 1$
$\hat{v}^{\pi}(s_1) = 0$	$\hat{v}^{\pi}(s_2) = 0$	$\hat{v}^{\pi}(s_3) = 0$	$\hat{v}^{\pi}(s_4) = 0$	$\hat{v}^{\pi}(s_5) = 0$

Monte Carlo(MC)

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

$$\hat{v}^{\pi}(S_t) \leftarrow \hat{v}^{\pi}(S_t) + \frac{1}{N(S_t)}(G_t - \hat{v}^{\pi}(S_t))$$

Suppose $\gamma = 0.95$

$$G_1 = 0 + 0 * 0.95 + (-1) * 0.95^2 + 1 * 0.95^3 = -0.045125$$

$G_2 = -0.0475$ $\quad$ $G_3 = -0.05$ $\quad$ $G_4 = 1$

$$\hat{v}^{\pi}(s_1) = 0 + \frac{1}{1} * (-0.045125 - 0) = -0.045125$$

$\hat{v}^{\pi}(s_2) = -0.0475$ $\quad$ $\hat{v}^{\pi}(s_3) = -0.05$ $\quad$ $\hat{v}^{\pi}(s_4) = 1$

- Repeat to converge
- May with exploring start or ϵ-greedy exploration

Monte Carlo & Temporal Difference

Temporal Difference(TD)

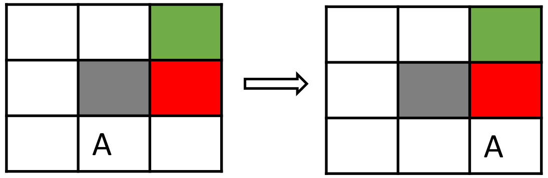
$$\hat{v}^{\pi}(S_t) \leftarrow \hat{v}^{\pi}(S_t) + \alpha\left(R_{t+1} + \gamma\hat{v}^{\pi}(S_{t+1}) - \hat{v}^{\pi}(S_t)\right)$$

Suppose $\gamma = 0.95, \alpha = 0.5$

TD target= $R_{t+1} + \gamma\hat{v}^{\pi}(S_{t+1})$

- Also need ϵ-greedy exploration
- Can count *n* step to construct *n*-step TD (if *n* tends to infinite, it become MC)

Step 2



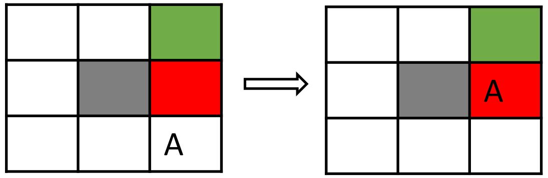
$R_3 = 0$ $\quad$ $R_4 = 0$

$\hat{v}^{\pi}(s_3) = 0$ $\quad$ $\hat{v}^{\pi}(s_4) = 0$

$$TD\ target = 0 + 0.95 * 0 = 0$$

$$\hat{v}^{\pi}(s_3) = 0 + 0.5 * (0 - 0) = 0$$

Step 3



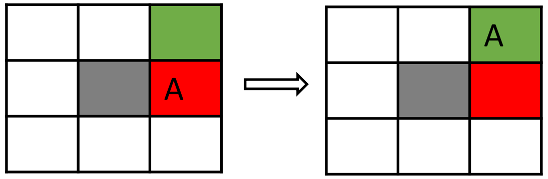
$R_3 = 0$ $\quad$ $R_4 = -1$

$\hat{v}^{\pi}(s_3) = 0$ $\quad$ $\hat{v}^{\pi}(s_4) = 0$

$$TD\ target = -1 + 0.95 * 0 = -1$$

$$\hat{v}^{\pi}(s_3) = 0 + 0.5 * (-1 - 0) = -0.5$$

Step 4



$R_3 = -1$ $\quad$ $R_4 = 1$

$\hat{v}^{\pi}(s_3) = 0$ $\quad$ $\hat{v}^{\pi}(s_4) = 0$

$$TD\ target = 1 + 0.95 * 0 = 1$$

$$\hat{v}^{\pi}(s_3) = 0 + 0.5 * (1 - 0) = 0.5$$

Famous RL: Deep Q-nerwork

This is a famous model-free RL algorithm, and lots of advanced methods inspired by it, paper is here.

But somehow it is not complex, mainly CNN + Q-learning

Review **Q-learning**

$$\hat{q}^{\pi}(S_t, A_t) \leftarrow \hat{q}^{\pi}(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a} \hat{q}^{\pi}(S_{t+1}, a) - \hat{q}^{\pi}(S_t, A_t)\right)$$

A TD algorithm, value iteration on q

Why need CNN?
For most realistic games, raw state is image, use only q-learning always need hand-crafted feature (or may have dimension curse), but hard to know what the best feature is.
With CNN, can realize end-to-end policy which is more like human-level control, and can use strong representational ability of deep network which can learn from data.

Famous RL: Deep Q-nerwork

$$\hat{q}^{\pi}(S_t, A_t) \leftarrow \hat{q}^{\pi}(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a} \hat{q}^{\pi}(S_{t+1}, a\ ) - \hat{q}^{\pi}(S_t, A_t) \right)$$

The only thing is Q function, so use deep network to approximate it.

$$Q(s, a) \approx f(s, a, w)$$

f is our network, w is its parameters.

So-called **Value Function Approximation**

Consider low dimensionality of action compared with observation and saving computation, network like:

s (raw image) → $\boxed{Conv_1}$ → …… → $\boxed{fc_n}$ → $\boxed{Q_{(s,a_1)}}$ …… $\boxed{Q_{(s,a_n)}}$

Then how to train to make it fit the real Q function?
Construct loss based on Q-learning

Famous RL: Deep Q-nerwork

$$\hat{q}^{\pi}(S_t, A_t) \leftarrow \hat{q}^{\pi}(S_t, A_t) + \alpha \left(\underbrace{R_{t+1} + \gamma \max_a \hat{q}^{\pi}(S_{t+1}, a)}_{\text{TD target}} - \hat{q}^{\pi}(S_t, A_t) \right)$$

$$\Downarrow$$

DQN loss function: $L(w) = \mathbb{E}[(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{Target} - Q(s, a, w))^2]$

- Q is the network
- w is the trainable parameters
- s' is the next state

Now achieve most intuitive main framework, but apply it to real environment always meet lots of problems, like unstable training.
So DQN in fact contain something more.

Famous RL: Deep Q-nerwork

Some problems:

- For DL, we always assume that samples are independently and identically distributed.
- But for RL, samples derived from interactions between policy and environment, sequence states are dependent.
- During the interaction between policy and environment, sample distribution of RL always changes.
- Previous works show that using non-linear network to approximate value function is always unstable

To rescue:

- DQN use experience replay to ease sample's iid problem
- DQN use two networks fit current Q function and target Q function respectively to smooth Q function approximation

Experience replay

Store historical data, when training, sample data from all include stored history

Famous RL: Deep Q-nerwork

DQN now can work but sometimes unstable and need carefully adjust hyperparameters.

Some further developments include Double DQN, Prioritized Replay, Dueling Network and so on.

DQN belongs to value-based RL algorithm, while some other RL algorithms are pretty different, which directly formulate the policy, like TRPO, PPO and so on.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Whole algorithm for DQN

Model-based RL: MBMF

Nowadays people mainly focus on model-free RL, which is powerful but always need plenty of samples. Model-based RL should be more sample-efficient, but have lots of imitation and hard to extend to complex problem and complex model.

MBMF is a model-based RL algorithm.

Maybe combine model-free and model-based RL in some tasks is a promising way, like this develop continuous DQN with model-based acceleration, and MBMF also refer to use model-based policy to initialize model-free learner.

Model-based RL: MBMF

MBMF focus on MuJoCo (locomotion tasks), so continuous observation and action, and this work assumes to access underlying reward function (can encode different tasks)

For model-based RL, a model of the dynamics is used to make predictions, which is used for action selection.

$\hat{f}_\theta(s_t, a_t)$ denotes a learned discrete-time dynamics function, parameterized by θ, output an estimation of the next state at time $t + \Delta t$ in, for example, a neural network.

Then solve optimizing problem: $$(\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}) = \arg \max_{\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}} \sum_{t'=t}^{t+H-1} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

Model-based RL: MBMF

Considering s_t and s_{t+1} may be too similar in continuous control problem, and action may has seemingly little effect on the output, a function directly map from (s_t, a_t) to s_{t+1} may hard to learn.
MBMF instead learns a dynamics function map from (s_t, a_t) to Δs, which means $\hat{s}_{t+1} = s_t + \hat{f}_\theta(s_t, a_t)$.

Then we need to train the dynamics function.
Collecting training data **D**, random policy interact with environment to produce trajectories is ok.
Construct error to minimize to train $\hat{f}_\theta(s_t, a_t)$

$$\mathcal{E}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \frac{1}{2} \|(\mathbf{s}_{t+1} - \mathbf{s}_t) - \hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)\|^2$$

Model-based RL: MBMF

After having a learned model $\hat{f}_\theta(s_t, a_t)$, utilize it is not so straightforward, since solving this is always difficult:

$$\mathbf{A}_t^{(H)} = \arg\max_{\mathbf{A}_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \quad :$$

$$\hat{\mathbf{s}}_t = \mathbf{s}_t, \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + \hat{f}_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}).$$

MBMF use **random-sampling shooting method**, simple method sufficient to simple problem.

- This method randomly generate K action sequences.
- Then can get corresponding states based on our previous learned dynamics model
- And the rewards for all sequences can be calculated
- Finally choose the candidate action sequence with highest expected cumulative reward

However, rather then having the policy execute this action sequence in open-loop, MBMF use **model predictive control (MPC):**
For a chosen action sequence, only executes the first action a_t, so receives updated state information s_{t+1}, and recalculates the optimal action sequence at the next time step.

Model-based RL: MBMF

Also use **on-policy data aggregation (in red block)** to improve performance by mitigating the mismatch between the data's state-action distribution and the model-based controller's distribution.

Algorithm 1 Model-based Reinforcement Learning

gather dataset $\mathcal{D}_{\text{RAND}}$ of random trajectories
initialize empty dataset $\mathcal{D}_{\text{RL}}$, and randomly initialize $\hat{f}_\theta$
for iter=1 **to** max_iter **do**
    train $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$ by performing gradient descent on Eqn. 2, using $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$
    for $t = 1$ **to** T **do**
        get agent's current state $\mathbf{s}_t$
        use $\hat{f}_\theta$ to estimate optimal action sequence $\mathbf{A}_t^{(H)}$ (Eqn. 4)
        execute first action $\mathbf{a}_t$ from selected action sequence $\mathbf{A}_t^{(H)}$
        add $(\mathbf{s}_t, \mathbf{a}_t)$ to $\mathcal{D}_{\text{RL}}$
    end for
end for

$$\mathcal{E}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \frac{1}{2} \|(\mathbf{s}_{t+1} - \mathbf{s}_t) - \hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)\|^2 \quad (2)$$

$$\mathbf{A}_t^{(H)} = \arg\max_{\mathbf{A}_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \quad : \quad \hat{\mathbf{s}}_t = \mathbf{s}_t, \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + \hat{f}_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}). \quad (4)$$

Exercise

- Play with this [web]() to see Dynamic programming(DP) and Temporary Difference(TD) on GridWord
- Test MC, TD, sarse and q-learning on FrozenLake-v0 (Gym). Code can be found on Github, also not hard to implement.