Day 7. Model-based RL

NPEX Reinforcement Learning

2020.09.04

Jaeuk Shin, Minkyu Park

CORE
Control + Optimization Research Lab

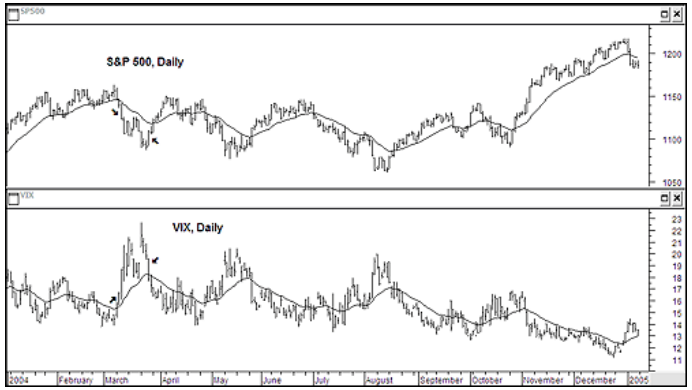
Model-based RL - Review

So far, we have learned **model-free** RL algorithms,
i.e., learning policy/value function were done without model info:

$$s_{t+1} \sim p(\cdot|s_t, a_t), \quad r_t = r(s_t, a_t).$$

What is p?

$\longrightarrow$ law of physics, artificial rules, etc.



Model-Based RL - Review

Model-based RL - Review

Can we **learn** p?

simplest case : $s_{t+1} = s_t + f(s_t, a_t)$

Assume we have a large number of transition samples (s_j, a_j, s'_j) from the **true** transition dynamics f.

Then, we may learn a parametrized model f_θ which is **close** to f.

How?

Model-based RL - Review

Given a batch $B = \{(s_j, a_j, s'_j)\}_{j=1}^N$, we construct a loss as follows:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{j=1}^{N} \|s_{t+1} - s_t - f_\theta(s_t, a_t)\|^2$$

and update θ by gradient-based algorithms.

What can we do if we have a good model?

Model-based RL - Review

One can apply some well-known methods in control theory, such as **model predictive control(MPC)**:

$$\max_{A_t^{(H)}=(a_t,\cdots a_{t+H-1})} \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, a_{t'})$$

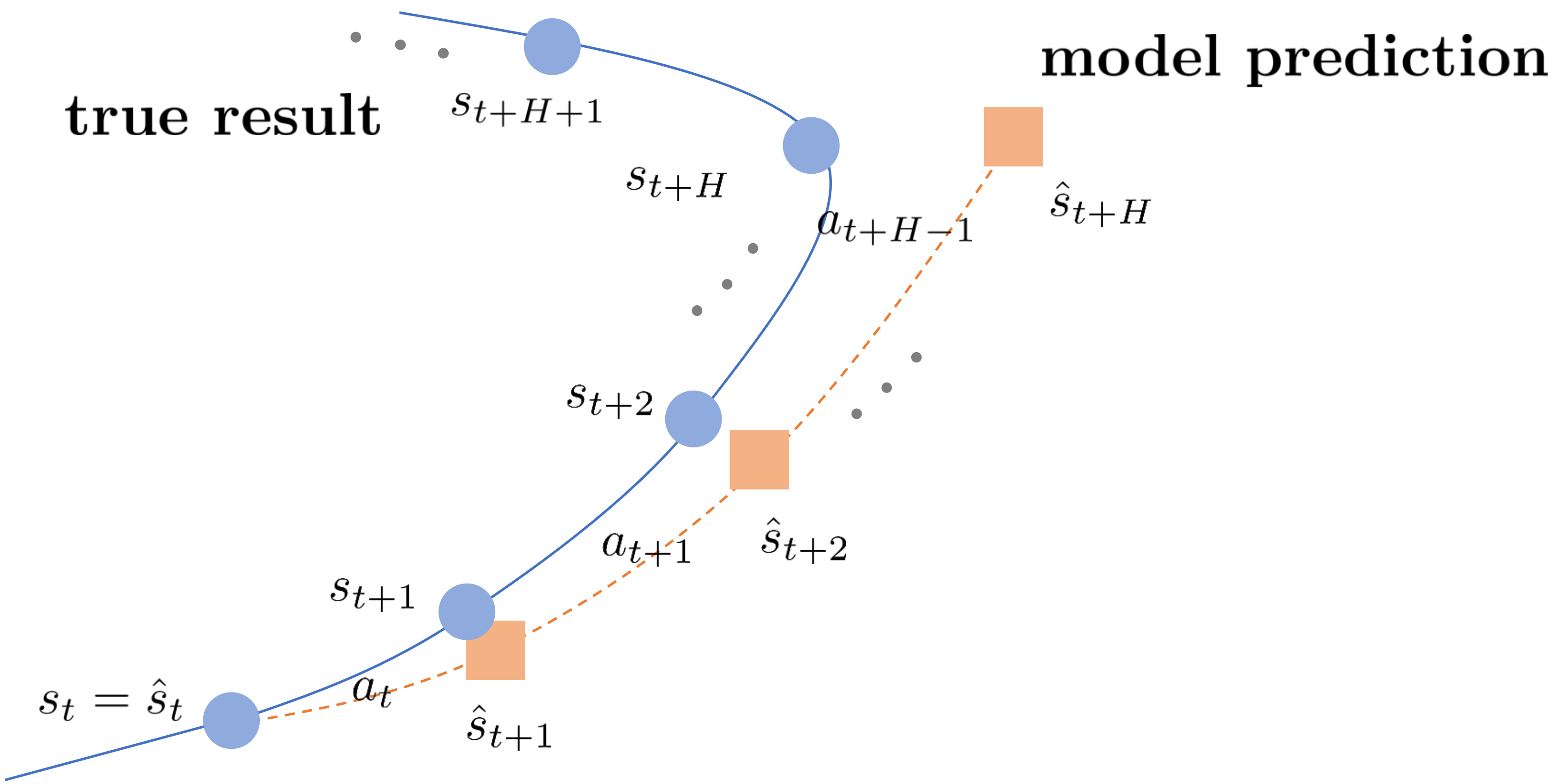
where

$$\hat{s}_t = s_t,$$

$$\hat{s}_{k+1} = \hat{s}_k + f_\theta(\hat{s}_k, a_k), \quad k = t, \cdots t+H-1.$$

In this case, we will use a simple algorithm so called **random sampling shooting method**.

Model-based RL - Review



Model-based RL - Review

Random sampling shooting is...

1. Generate N action sequences $\left\{\left\langle a_t^{(i)}, \cdots, a_{t+H-1}^{(i)}\right\rangle : 1 \leq i \leq N\right\}$ randomly.

2. For each action sequence, compute the predicted state trajectory $\left\langle \hat{s}_t^{(i)}, \cdots, \hat{s}_{t+H-1}^{(i)}\right\rangle$ and the resulting cost $c^{(i)} := \sum_{k=t}^{t+H-1} c\left(\hat{s}_k^{(i)}, a_k^{(i)}\right)$.

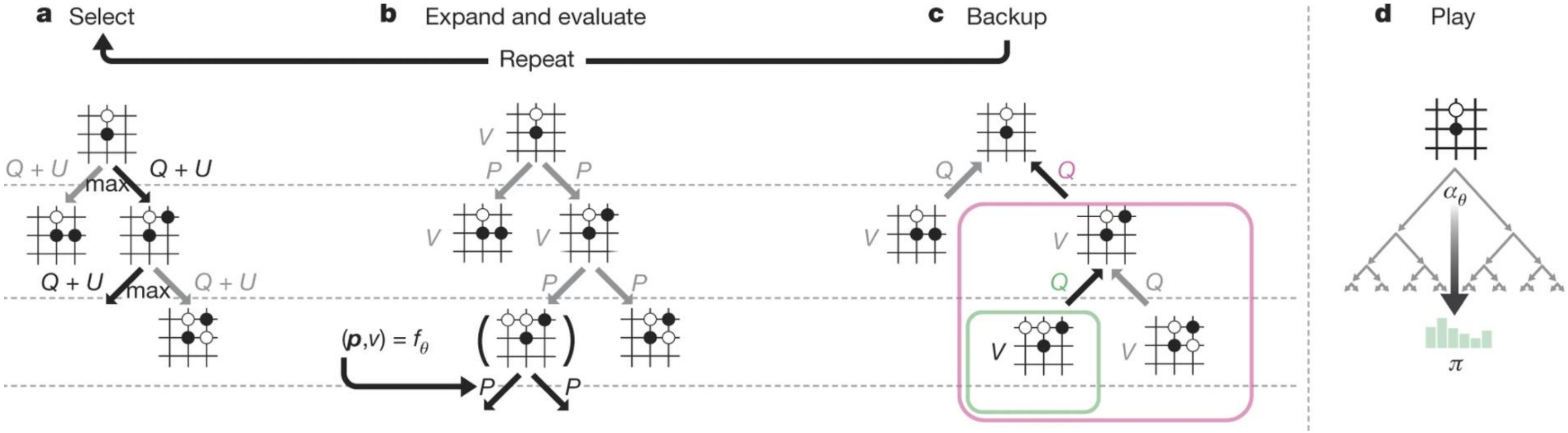
3. Choose the action sequence with largest $c^{(i)}$ as a solution.

other options? **cross-entropy method**(included in the practice session code), tree search, path integral optimal control, etc.

Model-based RL - Review

other options?

$\longrightarrow$ cross entropy method, tree search, etc.



Model-based RL - Review

Summary:

Algorithm 1 Model-based Reinforcement Learning

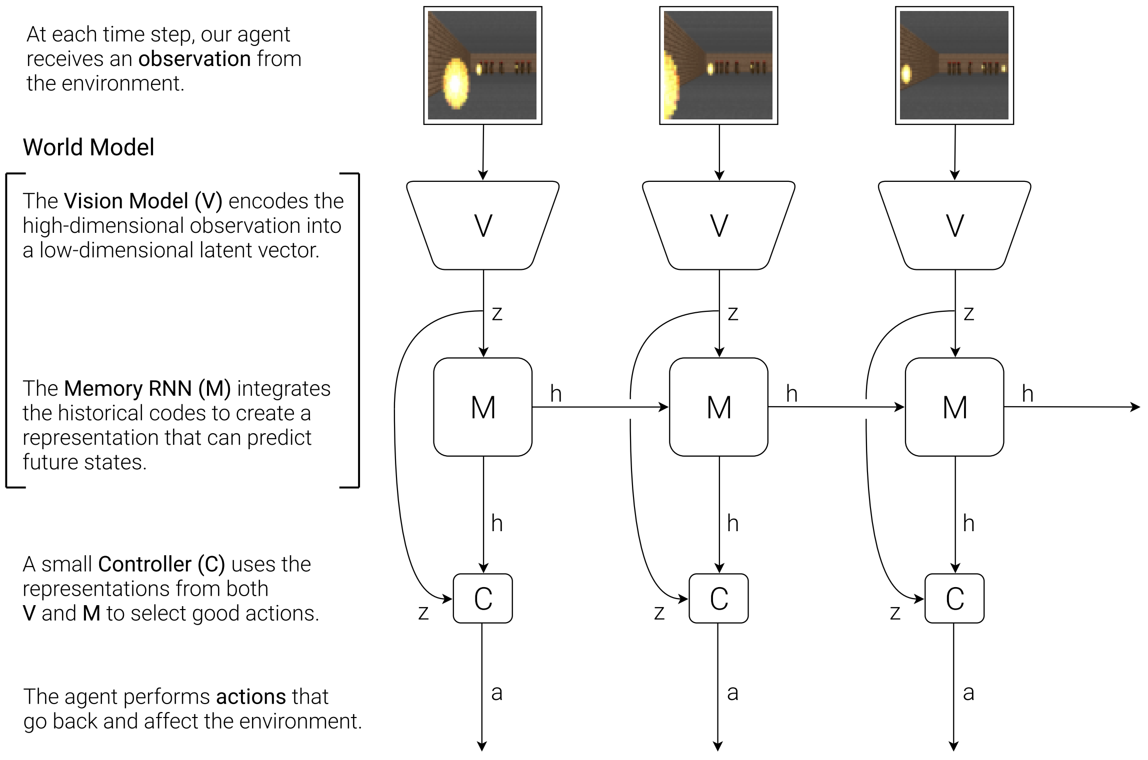
```
gather dataset $\mathcal{D}_{\text{RAND}}$ of random trajectories
initialize empty dataset $\mathcal{D}_{\text{RL}}$, and randomly initialize $\hat{f}_\theta$
for iter=1 to max_iter do
    train $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$ by performing gradient descent on Eqn. 2,
    using $\mathcal{D}_{\text{RAND}}$ 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
```

so simple!

Model-based RL - Review

more advanced Model-based RL algorithms?

Ex) World Models (Ha, Schmidhuber, 2018)



https://worldmodels.github.io/

Model-based RL - Implementation

Model-based RL - Implementation

```python
class TransitionMemory:
    def __init__(self, state_dim, act_dim, maxlen=20000):
        ...
    def append(self, state, act, next_state):
        self.data.append((state, act, next_state))

    def sample_batch(self, size):
        ...
        return (state_batch, act_batch, next_state_batch)
```

neural network : trained in supervised manner

$\Longrightarrow$ only (s, a, s') needed! (**known reward function**)

Model-based RL - Implementation

```
class TransitionModel(nn.Module):
    def __init__(self, state_dim, act_dim, hidden1, hidden2):
        super(TransitionModel, self).__init__()
        self.state_dim = state_dim
        self.act_dim = act_dim
        self.fc1 = nn.Linear(state_dim + act_dim, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, state_dim)

    def forward(self, state, act):
        x = torch.cat([state, act], dim=1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        delta = self.fc3(x)
        next_state = state + delta

        return next_state
```

neural network approximation of deterministic dynamics

Q. What if the transitions are stochastic?

Model-based RL - Implementation

```python
class ModelBasedAgent:
    def __init__(self, ...):
        ...

    def train(self, batch_size):
        self.model.train()
        (state_batch, act_batch, next_state_batch) = self.memory.sample_batch(batch_size)
        state_batch = torch.tensor(state_batch).float()
        act_batch = torch.tensor(act_batch).float()
        next_state_batch = torch.tensor(next_state_batch).float()

        prediction = self.model(state_batch, act_batch)
        loss_ftn = MSELoss()
        loss = loss_ftn(prediction, next_state_batch)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        loss_val = loss.detach().numpy()
        return loss_val
```

Remark. This is a supervised learning, so we may try to measure validation error!

Model-based RL - Implementation

```
def solve_random_shooting_opt(self, state, H, N):
    scores = np.zeros(N)
    action_sequences = self.ctrl_range * (2. * np.random.rand(H, N, dimA) - 1.)

    states = np.tile(state, (N, 1))
    for t in range(H):
        actions = action_sequences[t]
        scores += self.reward_model(states, actions)
        s = torch.tensor(states).float()
        a = torch.tensor(actions).float()
        next_s = self.model(s, a)
        states = next_s.detach().numpy()

    best_seq = np.argmax(scores)

    return action_sequences[best_seq]
```

generation of multiple control trajectories at once

simultaneous computation of predictions & cost along paths

CORE
Control + Optimization Research Lab

Model-based RL - Implementation

Some observations on random-shooting method

- Increasing # of trajectories : relatively less burden
- Increasing **horizon length** : **critical** to speed!
  - small horizon length $\Longrightarrow$ low quality of resulting control
  - long horizon length $\Longrightarrow$ high computational burden

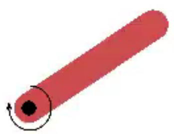
In general, MPC is a computational bottleneck in complex, high-dimensional domains.

Model-based RL - Experiment

Model-based RL – Experiment

OpenAI Gym Pendulum-v0

- environment file modified to allow simultaneous evalution of multiple cost functions
- Test both random-shooting method & cross-entropy method!



CORE
Control + Optimization Research Lab

Thank you