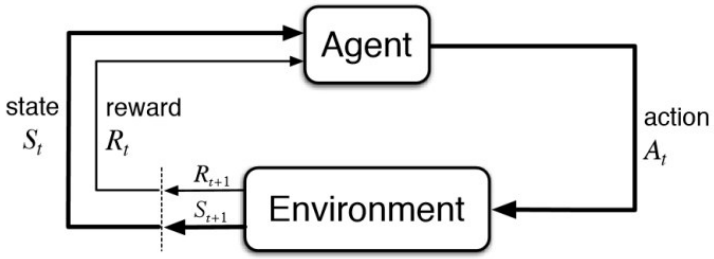
Deep Reinforcement Learning Basic

Dongda Li
dli160@syr.edu

Contents

1 Basic

1.1 Markov Decision Process (MDP)



The MDP consists of the following components: environment, state, observation, reward, action, and agent.

1.2 Policy

$$\pi : S \times A \to [0, 1] \quad \text{(Equation ??)}$$
$$\pi(a \mid s) = P(a_t = a \mid s_t = s)$$

1.3 State-value Function

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \quad \text{(Equation ??)}$$

$$V_\pi(s) = E[R] = E\Big[\sum_{t=0}^{\infty} \gamma^t r_t \;\Big|\; s_0 = s\Big]$$

where r_t is the reward at step t and $\gamma \in [0, 1]$ is the discount rate.

1.4 Value Function

$$V^\pi(s) = E[R \mid s, \pi] \quad \text{(Equation ??)}$$
$$V^*(s) = \max_\pi V^\pi(s)$$

1.5 Action-Value Function

$$Q^\pi(s, a) = E[R \mid s, a, \pi] \quad \text{(Equation ??)}$$
$$Q^*(s) = \max_a Q^\pi(s, a)$$

1.6 Method Classification

- **Model-based:** Use previous observations to *predict* following rewards and observations.
- **Model-free:** Train directly by experience.
- **Policy-based:** Directly approximate the policy of the agent.
- **Value-based:** The agent calculates the *value* of every possible action.
- **Off-policy:** The ability of the method to learn from old *historical data*.
- **On-policy:** Requires *fresh data* obtained from the environment.

1.7 Policy-based Method

This method is treated much like a classification problem:

- **NN Input:** Observation.
- **NN Output:** Distribution of actions.
- **Agent:** Randomly choose an action based on the action distribution (policy).

2 Cross-Entropy Method

2.1 Steps

1. Play N episodes using our current model and environment.
2. Calculate the total reward for every episode and decide on a reward boundary (usually a percentile, e.g. 50th or 70th).
3. Discard all episodes with a reward below the boundary.
4. Train on the remaining "elite" episodes using observations as input and the corresponding actions as the desired output.
5. Repeat from step 1 until the result is satisfactory.

Use the **cross-entropy loss** function as the loss function.

Drawback: Cross-entropy methods make it difficult to understand which step or state is good or not, as they only indicate that an entire episode is better overall.

3 Tabular Learning

3.1 Why Use Q but Not V?

If I know the value $V(s)$ of the current state, I know whether the state is good or not, but I do not know how to choose the next action. Even if I know the V of all subsequent states, I *cannot directly* determine the required action. Therefore, we decide the action based on Q.

If I know the Q-values of all available actions, we simply choose the action with the maximum Q. According to the relationship between Q and V, this action will also lead to the maximum V.

3.2 The Value Iteration in an Environment with a Loop

If there is no discount factor γ (i.e., $\gamma = 1$) and the environment has a loop, the value of the state becomes infinite.

3.3 Problems in Q-learning

- The state space may be non-discrete.
- The state space may be very large.
- The probability and reward matrices $P(s', r \mid s, a)$ may be unknown.

3.4 Value Iteration

3.4.1 Reward Table

- **Index:** "source state" + "action" + "target state".
- **Value:** Reward.

3.4.2 Transition Table

- **Index:** "state" + "action".
- **Value:** A table where the index is the next state and the value is the count.

3.4.3 Value Table

- **Index:** State.
- **Value:** Value of the state.

3.4.4 Steps

1. Use random actions to build the reward and transition tables.
2. Perform a value iteration loop over all states.
3. Play several full episodes to choose the best action using the updated value table while updating the reward and transition tables with new data.

Problem of Separating Training and Testing: When training and testing are separated, if the task is difficult, random actions may fail to reach the final state, resulting in missing states that are near the final step. Therefore, training and testing may need to be conducted simultaneously, with some exploitation added during testing.

3.5 Q-learning

Different from value iteration, Q-learning updates a Q-value table:

3.5.1 Q-value Table

- **Index:** "state" + "action".
- **Value:** The action value (Q-value).

Here:

$$V(s) = \arg\max_a Q(s, a)$$

4 Deep Q-learning

4.1 DQN

- **Input:** State.
- **Output:** Values for all actions (for n actions) given the input state.
- **Classification:** Off-policy, value-based, and model-free.

4.2 Problems

- Balancing exploration and exploitation.
- Data is not independent and identically distributed (i.i.d.), which is required for neural network training.
- The MDP might be partially observable (POMDP).

4.3 Basic Tricks in DeepMind 2015 Paper

- Use ϵ-greedy to handle exploration and exploitation.
- Use a replay buffer and target network to enforce i.i.d.-like properties:
  - The replay buffer randomly selects experiences.
  - The target network isolates the influence of nearby Q-values during training.
- Use several observations as the state to handle POMDPs.

4.4 Double DQN

Idea: Choose actions for the next state using the trained network but use Q-values from the target network.

4.5 Noisy Networks

Idea: Add noise to the weights of fully-connected layers of the network and adjust the noise parameters during training via back-propagation. This replaces ϵ-greedy and improves performance.

4.6 Prioritized Replay Buffer

Idea: Improve sample efficiency by prioritizing experiences in the replay buffer according to the training loss.

Trick: Use loss weights to compensate for the distribution bias introduced by prioritization.

4.7 Dueling DQN

Idea: Decompose the Q-value $Q(s, a)$ into the state value $V(s)$ and the advantage $A(s, a)$.

Trick: Force the mean value of the advantage for any state to be zero.

4.8 Categorical DQN

Idea: Train the probability distribution of the action Q-values rather than a single Q-value.

Tricks:

- Use a generic parametric distribution, i.e., a fixed number of "atoms" placed regularly over a value range.
- Use the Kullback-Leibler (KL) divergence as the loss.

5 Policy Gradients

5.1 REINFORCE

5.1.1 Idea

Policy Gradient:

$$\Delta J \approx E\Big[Q(s, a)\, \Delta \log \pi(a \mid s)\Big] \quad \text{(Equation ??)}$$

Loss function:

$$\text{loss} = -Q(s, a)\, \log \pi(a \mid s) \quad \text{(Equation ??)}$$

Increase the probability of actions that yield high total reward and decrease the probability of actions that yield low total reward.

$$\pi(a \mid s) > 0, \quad -\log \pi(a \mid s) > 0 \quad \text{(Equation ??)}$$

5.1.2 Problems

- Requires full episodes to obtain Q from completed episodes.
- High variance gradients; long episodes tend to have larger Q than short ones.
- Convergence to locally optimal policies due to lack of exploration.
- Samples are not i.i.d.

5.1.3 Basic Tricks

- Use Actor-Critic methods to learn Q.
- Subtract a baseline from Q to reduce variance.
- Add an entropy term to the loss to encourage exploration.
- Use parallel environments to reduce sample correlation.

5.2 Actor-Critic

$$Q(s,a) = \sum_{i=0}^{N-1} \gamma^i r_i + \gamma^N V(s_N)$$
$$\text{Loss}_{\text{value}} = MSE\Big(V(s), Q(s,a)\Big) \quad \text{(Equation ??)}$$

$$Q(s,a) = A(s,a) + V(s)$$
$$\text{Loss}_{\text{policy}} = -A(s,a)\log\pi(a \mid s) \quad \text{(Equation ??)}$$

Here, $A(s,a)$ is the advantage. We train the critic using Equation **??** and the policy using Equation **??**. This approach is known as Advantage Actor-Critic (A2C).

Idea: The gradient's magnitude is proportional to the advantage $A(s,a)$. A separate network estimates $V(s)$ for each observation.

5.2.1 Implementation

In practice, the policy and value networks often share a common body and have separate output heads, which improves efficiency and convergence.

5.2.2 Tricks

- Add an entropy bonus to the loss:

$$H_{\text{entropy}} = -\sum \pi \log \pi, \quad \text{Loss}_{\text{entropy}} = \beta \sum_i \pi_\theta(s_i)\log\pi_\theta(s_i) \quad \text{(Equation ??)}$$

- Use multiple environments to improve stability.
- Apply gradient clipping to prevent large updates.

Total Loss Function:

$$\text{Loss} = \text{Loss}_{\text{policy}} + \text{Loss}_{\text{value}} + \text{Loss}_{\text{entropy}}.$$

5.2.3 Asynchronous Advantage Actor-Critic (A3C)

A3C uses parallel environments to speed up training. (Refer to open source implementations for further details.)

6 Deep Reinforcement Learning in Natural Language Processing (NLP)

6.1 Basic Concepts in NLP

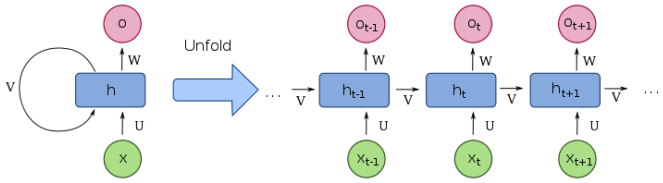
- Recurrent Neural Networks (RNNs).
- Word embeddings.
- The **seq2seq** model.
- Recurrent Models of Visual Attention (NIPS 2014).

See CS224d for more details in NLP.

6.2 RNNs

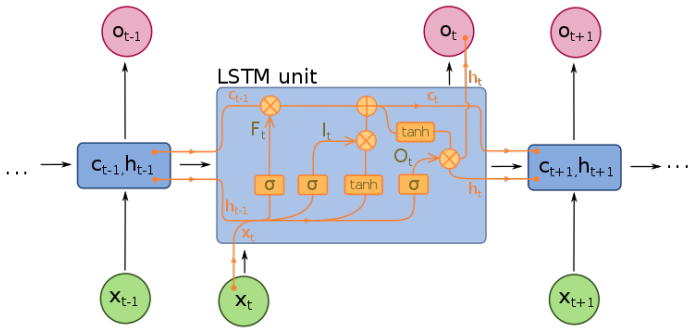
An RNN is a network that processes sequences by maintaining a hidden state (a vector) that is updated at each time step.

Unfolded RNN:



RNNs produce different outputs for the same input in different contexts and are standard building blocks for processing variable-length sequences.

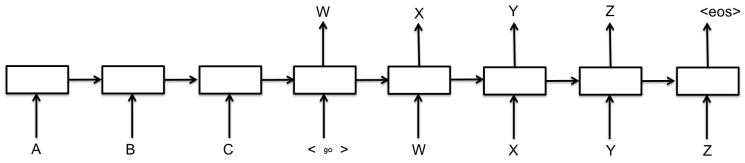
6.2.1 LSTM



6.3 Word Embedding (word2vec)

Word embeddings map words or phrases to vectors of real numbers by embedding a one-hot vector (with one dimension per word) into a lower-dimensional continuous space. These embeddings are useful for tasks such as syntactic parsing and sentiment analysis. See Word Embedding for details.

6.4 Encoder-Decoder (Seq2Seq)

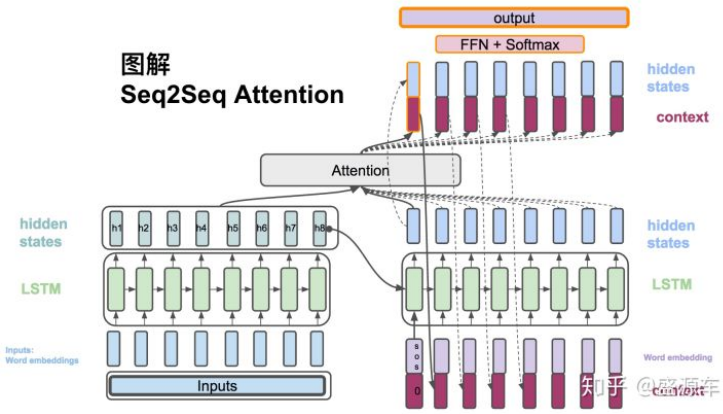


An encoder RNN processes an input sequence into a fixed-length representation, which is then passed to a decoder RNN that produces an output sequence. This framework is widely used in machine translation.

Modes:

- **Teacher-forcing mode:** The decoder input is the target reference.
- **Curriculum learning mode:** The decoder input is the previous output of the decoder.

Attention Mechanism: Improves the performance of seq2seq models by allowing the decoder to focus on different parts of the input sequence. For example:



(The attention mechanism figure is from Zhihu.)

6.5 RL in Seq2Seq

- Sample from the probability distribution rather than using an average result.
- When the score is not differentiable, policy gradients can be used with the score as a scaling factor.
- Stochasticity is introduced in the decoding process when the dataset is limited.
- The argmax score can be used as a baseline for Q-values.

7 Continuous Action Space

8 DDPG

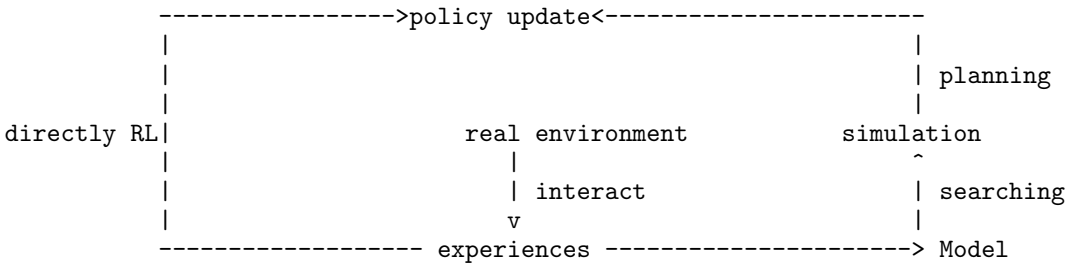
TBC.

9 Model-based RL

9.1 What is Model-based?

Consider that our MDP is a model that takes an input state s and action a and outputs a reward r and new state s'; i.e., $(r, s') = M(s, a)$. Sometimes we do not know the exact model of the MDP, but we can learn it from experience. The model is used for search and planning: search can generate additional simulated experience, and planning uses simulated experience to update our policy and value function. Planning methods can be similar to those used in model-free RL.

Diagram:



Why use model-based RL? We want to obtain more experience through simulation and thereby speed up learning.

9.2 How to Search?

To generate additional experience, we can perform search methods such as:

- Rollout search.
- Monte Carlo Tree Search (MCTS).
- ...

10 NN Functions

10.1 Sigmoid

Description: It transforms a value input to the range $(0, 1)$.

$$f(x) = \frac{L}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

10.2 Softmax

Description: It transforms a K-dimensional vector input to a probability distribution over K classes (each entry in $(0, 1)$ and summing to 1).

10.3 Tanh

Description: It transforms an input value to the range $(-1, 1)$.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

10.4 ReLU

$$f(x) = \max(0, x)$$

11 Reference

- Maxim Lapan, *Deep Reinforcement Learning Hands-On*, 2018.
- Mnih, V., Kavukcuoglu, K., Silver, D., et al. *Human-level control through deep reinforcement learning*. Nature, 2015, 518(7540):529.
- Mnih, V., Heess, N., Graves, A. *Recurrent models of visual attention*. Advances in Neural Information Processing Systems, 2014:2204–2212.
- Paszke, Adam, Gross, et al. *Automatic Differentiation in PyTorch*, 2017.