Note: A Brief Survey of Deep Reinforcement Learning

一文弄懂神经网络中的反向传播法——BackPropagation

https://www.cnblogs.com/charlotte77/p/5629865.html

Introduction:

1. One of the primary goals of the field of artificial intelligence (AI) is to produce fully autonomous agents that interact with their environments to learn optimal behaviours, improving over time through trial and error.
2. 之前的RL 和AI lacked scalability（可拓展性）and were inherently limited to fairly low-dimensional problems. 是因为RL algorithms share the same complexity issues as other algorithms: memory complexity, computational complexity, and in the case of machine learning algorithms, sample complexity
3. The most important property of deep learning is that deep neural networks can automatically find compact low-dimensional representations (features) of high-dimensional data
4. “deep reinforcement learning” (DRL) 是用了deep learning 的 RL：with the use of deep learning algorithms within RL
5. 这个research的目的就是 cover both seminal（开创性的） and recent developments in DRL, conveying the innovative ways in which neural networks can be used to bring us closer towards developing autonomous agents.
6. 讲了DRL在很多领域都有用：robotics， 下象棋，video game等等。在未来，DRL will be an important component in constructing general AI systems

REWARD-DRIVEN BEHAVIOR

1. 什么是RL：
    a. The essence of RL is learning through interaction
        i. An RL agent interacts with its environment and, upon observing the consequences of its actions
    b. The other key influence on RL is optimal control
        i. lent the mathematical formalisms (most notably dynamic programming) that underpin（支撑）the field.
2. RL set-up:
    a. an autonomous agent, controlled by a machine learning algorithm, observes a state st from its environment at timestep t.
    b. The agent interacts with the environment by taking an action at in state st
    c. When the agent takes an action, the environment and the agent transition to a new state st+1 based on the current state and the chosen action.
3. Agent goal:
    a. to learn a policy (control strategy) π that maximises the expected return (cumulative, discounted reward)
    b. optimal policy
        i. is any policy that maximises the expected return in the environment.

c. the challenge in RL is that the agent needs to learn about the consequences of actions in the environment by trial and error, unlike in optimal control, a model of the state transition dynamics is not available to the agent.

4. Markov Decision Processes

A. *Markov Decision Processes*

Formally, RL can be described as a Markov decision process (MDP), which consists of:

- A set of states $\mathcal{S}$, plus a distribution of starting states $p(\mathbf{s}_0)$.
- A set of actions $\mathcal{A}$.
- Transition dynamics $\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that map a state-action pair at time t onto a distribution of states at time $t+1$.
- An immediate/instantaneous reward function $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.
- A discount factor $\gamma \in [0, 1]$, where lower values place more emphasis on immediate rewards.

a.

b. In general, the policy π is a mapping from states to a probability distribution over actions:: $\pi : \mathcal{S} \rightarrow p(\mathcal{A} = \mathrm{a}|\mathcal{S})$.

c. If MDP is episodic:

i. 返回值表示形式只在有限时间点条件（或者说有终止时间点）

ii. episodes: 决策体与环境的交互过程的子片段（任意重复性的交互过程），称为 episodes

iii. terminal state: 每个子片段都有一个特殊状态，其后续时间点被重置进入新片段，这样的特殊片段，称为 terminal state

iv. the state is reset after each episode of length T, then the sequence of states, actions and rewards in an episode constitutes a trajectory(弹道) or rollout(推出) of the policy

v. Return accumulates rewards $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$.

vi. Goal of RL is to find an optimal policy π∗ : $\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R|\pi]$.

d. If MDP is non-episodic

i. where T = ∞. In this situation, γ < 1 prevents an infinite sum of rewards from being accumulated.

e. A key concept underlying RL is the Markov property— only the current state affects the next state

i. unrealistic, as it requires the states to be fully observable.

f. partially observable MDPs (POMDPs)

i. the agent receives an observation ot ∈ Ω, where the distribution of the observation p(ot+1|st+1, at) is dependent on the current state and the previous action

5. Challenges in RL
   a. The optimal policy must be inferred by trial-and-error interaction with the environment. The only learning signal the agent receives is the reward.
   b. The observations of the agent depend on its actions and can contain strong temporal correlations.
   c. Agents must deal with long-range time dependencies: Often the consequences of an action only materialise after many transitions of the environment. This is known as the (temporal) credit assignment problem
6. REINFORCEMENT LEARNING ALGORITHMS
   a. There are two main approaches to solving RL problems: methods based on value functions and methods based on policy search. There is also a hybrid, actor-critic approach, which employs both value functions and policy search.
   b. Value Functions

i.

Value function methods are based on estimating the value (expected return) of being in a given state. The *state-value function* $V^\pi(\mathbf{s})$ is the expected return when starting in state $\mathbf{s}$ and following π henceforth:

$$V^\pi(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi] \tag{2}$$

ii.

The optimal policy, π^*, has a corresponding state-value function $V^*(\mathbf{s})$, and vice-versa, the optimal state-value function can be defined as

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}. \tag{3}$$

If we had $V^*(\mathbf{s})$ available, the optimal policy could be retrieved by choosing among all actions available at $\mathbf{s}_t$ and picking the action $\mathbf{a}$ that maximises $\mathbb{E}_{\mathbf{s}_{t+1}\sim\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a})}[V^*(\mathbf{s}_{t+1})]$.

In the RL setting, the transition dynamics $\mathcal{T}$ are unavailable. Therefore, we construct another function, the *state-action-value* or *quality function* $Q^\pi(\mathbf{s}, \mathbf{a})$, which is similar to V^π, except that the initial action $\mathbf{a}$ is provided, and π is only followed from the succeeding state onwards:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R|\mathbf{s}, \mathbf{a}, \pi]. \tag{4}$$

The best policy, given $Q^\pi(\mathbf{s}, \mathbf{a})$, can be found by choosing $\mathbf{a}$ greedily at every state: $\operatorname{argmax}_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$. Under this policy, we can also define $V^\pi(\mathbf{s})$ by maximising $Q^\pi(\mathbf{s}, \mathbf{a})$: $V^\pi(\mathbf{s}) = \max_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$.

Dynamic Programming: To actually learn Q^π, we exploit the Markov property and define the function as a Bellman equation [13], which has the following recursive form:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}}[r_{t+1} + \gamma Q^\pi(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))]. \quad (5)$$

This means that Q^π can be improved by *bootstrapping*, i.e., we can use the current values of our estimate of Q^π to improve our estimate. This is the foundation of Q-learning [159] and the state-action-reward-state-action (SARSA) algorithm [112]:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q^\pi(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta, \quad (6)$$

where α is the learning rate and $\delta = Y - Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ the temporal difference (TD) error; here, Y is a target as in a standard regression problem. SARSA, an *on-policy* learning algorithm, is used to improve the estimate of Q^π by using transitions generated by the behavioural policy (the policy derived from Q^π), which results in setting $Y = r_t + \gamma Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$. Q-learning is *off-policy*, as Q^π is instead updated by transitions that were not necessarily generated by the derived policy. Instead, Q-learning uses $Y = r_t + \gamma \max_{\mathbf{a}} Q^\pi(\mathbf{s}_{t+1}, \mathbf{a})$, which directly approximates Q^*.

iii.

To find Q^* from an arbitrary Q^π, we use *generalised policy iteration*, where policy iteration consists of *policy evaluation* and *policy improvement*. Policy evaluation improves the estimate of the value function, which can be achieved by minimising TD errors from trajectories experienced by following the policy. As the estimate improves, the policy can naturally be improved by choosing actions greedily based on the updated value function. Instead of performing these steps separately to convergence (as in policy iteration), generalised policy iteration allows for interleaving the steps, such that progress can be made more rapidly.

iv.

c. Sampling
   i. Monte Carlo methods estimate the expected return (2) from a state by averaging the return from multiple rollouts of a policy.
   ii. Pure Monte Carlo methods can also be applied in non-Markovian environments.
      1. They can only be used in episodic MDPs, as a rollout has to terminate for the return to be calculated.
   iii. It is possible to get the best of both methods by combining TD learning and Monte Carlo policy evaluation： TD(λ) algorithm
      1. Similarly to the discount factor, the λ in TD(λ) is used to interpolate between Monte Carlo evaluation and bootstrapping。
   iv. Advantage function Aπ (s, a)

1. $A\pi$ instead represents relative state-action values
2. Learning relative values is akin to removing a baseline or average level of a signal
3. $A\pi$ represents a relative advantage of actions through the simple relationship $A\pi = Q\pi - V\pi$

d. Policy Search
   i. Policy search methods do not need to maintain a value function model, but directly search for an optimal policy $\pi*$.
   ii. A parameterised policy $\pi\theta$ is chosen, whose parameters are updated to maximize the expected return $E[R|\theta]$ using either gradient-based or gradient-free optimisation
   iii. Policy Gradients
      1. Gradients can provide a strong learning signal as to how to improve a parameterised policy
      2. we need to average over plausible trajectories induced by the current policy parameterisation
         a. averaging requires either deterministic approximations (e.g., linearisation) or stochastic approximations via sampling
         b. Deterministic approximations can only be applied in a model-based setting where a model of the underlying transition dynamics is available
      3. In the more common model-free RL setting, a Monte Carlo estimate of the expected return is determined.
         a. this Monte Carlo approximation poses a challenge since gradients cannot pass through these samples of a stochastic function.
         b. Therefore, we turn to an estimator of the gradient, known in RL as the REINFORCE rule
            i. can be used to compute the gradient of an expectation over a function f of a random variable X with respect to parameters θ
            ii. $\nabla_\theta \mathbb{E}_X[f(X;\theta)] = \mathbb{E}_X[f(X;\theta)\nabla_\theta \log p(X)].$
            iii. the resulting gradients possess a high variance.
            iv. Thus we need to reduce the variance. The general methodology for performing this is to subtract a baseline
               1. The simplest baseline is the average return taken over several episodes
   iv. Actor-critic Methods:
      1. The “actor” (policy) learns by using feedback from the “critic” (value function).
      2. these methods trade off variance reduction of policy gradients with bias introduction from value function methods

3. Actor-critic methods use the value function as a baseline for policy gradients
4. difference between actor-critic methods and other baseline methods are that actor-critic methods utilize a learned value function

e. Planning and Learning
   i. Sutton and Barto [135] define planning as any method which utilizes a model to produce or improve a policy.
   ii. In RL, we focus on learning without access to the underlying model of the environment
      1. Model-free RL methods learn directly from interactions with the environment
      2. Model-based RL methods can simulate transitions using the learned model, resulting in increased sample efficiency
   iii. Learning a model introduces extra complexities, and there is always the danger of suffering from model errors
      1. use model predictive control, where planning is repeated after small sequences of actions in the real environment

f. The Rise of DRL
   i. DRL can deal efficiently with the curse of dimensionality
   ii. In general, DRL is based on training deep neural networks to approximate the optimal policy $\pi*$, and/or the optimal value functions $V*$, $Q*$ and $A*$.
   iii. Although there have been DRL successes with gradient free methods [37, 23, 64], the vast majority of current works rely on gradients
      1. gradients provide a strong learning signal.
   iv. Backpropagation
      1. benefit of backpropagation is to view the optimisation of the expected return as the optimisation of a stochastic function
      2. This function can comprise of several parts—models, policies and value functions—which can be combined in various ways
      3. it is possible to forward propagate and backpropagate through entire rollouts; on the other hand, inaccuracies can accumulate over long time steps
         a. it may be be pertinent to instead use a value function to summarize the statistics of the rollouts

7. Value Functions
   a. Function Approximation and the DQN
      i.

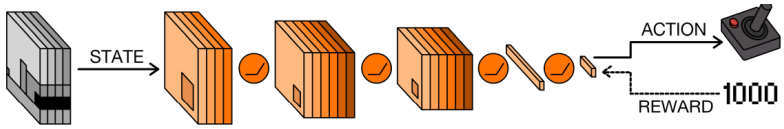

Fig. 5. The deep Q-network [84]. The network takes the state—a stack of greyscale frames from the video game—and processes it with convolutional and fully connected layers, with ReLU nonlinearities in between each layer. At the final layer, the network outputs a discrete action, which corresponds to one of the possible control inputs for the game. Given the current state and chosen action, the game returns a new score. The DQN uses the reward—the difference between the new score and the previous one—to learn from its decision. More precisely, the reward is used to update its estimate of Q, and the error between its previous estimate and its new estimate is backpropagated through the network.

ii. The inputs to the DQN are four greyscale frames of the game, concatenated over time, which are initially processed by several convolutional layers in order to extract spatiotemporal features
iii. The strength of the DQN lies in its ability to compactly represent both high-dimensional observations and the Q-function using deep neural networks
iv. The DQN addressed the fundamental instability problem of using function approximation in RL [145] by the use of two techniques: experience replay [80] and target networks
    1. Experience replay memory stores transitions of the form (st, at, st+1, rt+1) in a cyclic buffer, enabling the RL agent to sample from and train on previously observed data offline.
        a. Not only does this massively reduce the amount of interactions needed with the environment, but batches of experience can be sampled, reducing the variance of learning updates.
    2. Target network that initially contains the weights of the network enacting the policy, but is kept frozen for a large period of time. Rather than having to calculate the TD error based on its own rapidly fluctuating estimates of the Q-values, the policy network uses the fixed target network.

b. Q-Function Modifications
    i. the single estimator used in the Q-learning update rule overestimates the expected return due to the use of the maximum action value as an approximation of the maximum expected action value.
    ii. Double-Q learning provides a better estimate through the use of a double estimator
    iii. A more radical proposal by Bellemare et al. [12] was to actually learn the full value distribution, rather than just the expectation
        1. this provides additional information, such as whether the potential rewards come from a skewed or multimodal distribution
    iv. another way to adjust the DQN architecture is to decompose the Q-function into meaningful functions,
        1. constructing Q^{π} by adding together separate layers that compute the state-value function V^{π} and advantage function A^{π}
        2. the dueling DQN [157] benefits from a single baseline for the state in the form of V^{π} , and easier-tolearn relative values in the form of A^{π}
    v. The idea of using representation learning to create distributed embeddings is a particular strength of DRL,
    vi. in RL, when many actions need to be made simultaneously
        1. to factorize the policy, treating each action independently

2. An alternative is to construct an autoregressive policy, where each action in a single timestep is predicted conditionally on the state and previously chosen actions from the same timestep

8. Policy Search
    a. Policy search methods aim to directly find policies by means of gradient-free or gradient-based methods.
    b. gradient-free policy search algorithms
        i. eschewed the commonly used backpropagation algorithm in favor of evolutionary algorithms
            1. Evolutionary methods rely on evaluating the performance of a population of agents.
    c. Backpropagation through Stochastic Functions
        i. hard attention
            1. the stochastic variable would determine the coordinates of a small crop of the image, and hence reduce the amount of computation needed
            2. This usage of RL to make discrete, stochastic decisions over inputs
    d. Compounding Errors
        i. Searching directly for a policy represented by a neural network with very many parameters can be difficult and can suffer from severe local minima. Ways to around this problems:
            1. guided policy search (GPS)
                a. takes a few sequences of actions from another controller (which could be constructed using a separate method, such as optimal control).
                b. GPS learns from them by using supervised learning in combination with importance sampling, which corrects for off-policy samples
                c. GPS works in a loop, by optimizing policies to match sampled trajectories, and optimizing trajectory distributions to match the policy and minimize costs.
            2. trust region
                a. in which optimisation steps are restricted to lie within a region where the approximation of the true cost function still holds.
                b. the chance of a catastrophically bad update is lessened, and many algorithms that use trust regions guarantee or practically result in monotonic improvement in policy performance
                c. trust region policy optimisation (TRPO)
                    i. robust and applicable to domains with high-dimensional inputs

ii. TRPO optimizes a surrogate objective function—specifically, it optimizes an (importance sampled) advantage estimate, constrained using a quadratic approximation of the KL divergence.

d. generalized advantage estimation (GAE)
    i. The combination of TRPO and GAE remains one of the state of-the-art RL techniques in continuous control.
    ii. However, the constrained optimisation of TRPO requires calculating secondorder gradients, limiting its applicability

e. proximal policy optimisation (PPO)
    i. unconstrained optimisation, requiring only first-order gradient information

e. Actor-Critic Methods
    i. actor-critic approaches have grown in popularity as an effective means of combining the benefits of policy search methods with learned value functions, which are able to learn from full returns and/or TD errors.
    ii. One development in the context of actor-critic algorithms are deterministic policy gradients (DPGs):
        1. extend the standard policy gradient theorems for stochastic policies [164] to deterministic policies.
        2. DPGs only integrate over the state space, requiring fewer samples in problems with large action spaces
    iii. On-policy methods can be more stable, whilst off-policy methods can be more data efficient, and hence there have been several attempts to merge the two
        1. interpolated policy gradients (IPGs)
    iv. An orthogonal approach to speeding up learning is to exploit parallel computation.
        1. asynchronous advantage actor-critic (A3C)
            a. a framework for training multiple DQNs in parallel, achieving both better performance and a reduction in training time.
            b. A3C combines advantage updates with the actor-critic formulation, and relies on asynchronously updated policy and value function networks trained in parallel over several processing threads
                i. The use of multiple agents, situated in their own, independent environments, not only stabilizes improvements in the parameters, but conveys an additional benefit in allowing for more exploration to occur.
            c. advantage actor-critic (A2C)

i. segments from the trajectories of multiple agents can be collected and processed together in a batch, with batch processing more efficiently enabled by GPUs; synchronous version