Deep Q-Learning with Prioritized Sampling

Jianwei Zhai, Quan Liu(✉), Zongzhang Zhang, Shan Zhong, Haijun Zhu, Peng Zhang, and Cijia Sun

School of Computer Science and Technology, Soochow University, Suzhou 215000, China
quanliu@suda.edu.cn, 20144227023@stu.suda.edu.cn

Abstract. The combination of modern reinforcement learning and deep learning approaches brings significant breakthroughs to a variety of domains requiring both rich perception of high-dimensional sensory inputs and policy selection. A recent significant breakthrough in using deep neural networks as function approximators, termed Deep Q-Networks (DQN), proves to be very powerful for solving problems approaching real-world complexities such as Atari 2600 games. To remove temporal correlation between the observed transitions, DQN uses a sampling mechanism called experience reply which simply replays transitions at random from the memory buffer. However, such a mechanism does not exploit the importance of transitions in the memory buffer. In this paper, we use prioritized sampling into DQN as an alternative. Our experimental results demonstrate that DQN with prioritized sampling achieves a better performance, in terms of both average score and learning rate on four Atari 2600 games.

Keywords: Reinforcement Learning · Deep Learning · Deep Reinforcement Learning · Policy control · Prioritized sampling

1 Introduction

Recent breakthroughs in both modern reinforcement learning (RL) and deep learning (DL) have given rise to a new research direction called deep reinforcement learning (DRL) which combines advances in DL for sensory inputs processing with RL [1–3]. One of the most notable DRL methods called Deep Q-Networks (DQN) which combines a deep convolutional neural network with a variant Q-learning algorithm in RL, has been shown to be capable of learning control policies in complex environments with high-dimensional sensory inputs. DQN outperformed previous algorithms based on handcrafted features and achieved or even surpassed a level comparable to that of a skilled human player in some Atari 2600 games, using the same network architecture and hyperparameters [2].

On-line RL agents incrementally update the parameters of value functions when they encounter a sequence of highly correlated transitions [4]. However, most DL methods have two main requirements: (a) the training samples are

A. Hirose et al. (Eds.): ICONIP 2016, Part I, LNCS 9947, pp. 13–22, 2016.
DOI: 10.1007/978-3-319-46687-3_2

independent of each other; (b) the samples can be reused many times during training. A technique called experience replay [5] can be utilized to meet these requirements. Therefore, this sampling mechanism was applied to the DQN method [1,2], which stabilized the training of the algorithm. At each time step, the agent stores every observed transition into the memory buffer and then samples uniformly to get a number of mini-batch transitions for updating the parameters.

However, this approach of uniform sampling is in some respects limited because the memory buffer does not differentiate the importance of distinct transitions and always overwrites with recent transitions owing to the finite memory size [2]. So in this paper, we propose a novel sampling mechanism termed prioritized sampling which is more effective and efficient than the case of all transitions are sampled uniformly. Specifically, we set two priority levels on sampling transitions. On one hand, a more efficient sampling method should emphasize the transitions from which the agent can learn the most. And we all know that transitions with positive rewards are more informative and valuable for learning. So we assign these transitions with higher priority to be sampled during training. This modification of sampling can make transitions with positive rewards be sampled more frequently so as to learn optimal policies faster. On the other hand, we add an explicit penalty term to every transition being accessed to reduce the probability of being sampled again. The degrees of punishment will be higher with the increase of sampled times in order to make parts of the transitions never been sampled recently have chances to be reused in time. More importantly, we set the priority of sampling by measuring the magnitude of rewards higher than the latter.

This paper presents a new model-free, off-policy RL algorithm, called PS-DQN. PS-DQN makes two improvements based on the DQN algorithm. First, its network is trained with samples obtained by prioritized sampling to eliminate correlations between observed transitions. Second, it uses a soft target network to give consistent target Q-values during temporal difference backups. On four Atari 2600 games, PS-DQN appears better than DQN empirically, in terms of both average score and learning rate.

2 Background

2.1 Reinforcement Learning

In reinforcement learning, the agent interacts sequentially with an unknown environment, with the goal of maximizing cumulative rewards [4]. The environment is often formalized as a sequence of state transitions (s_t, a_t, r_t, s_t+1) of a Markov Decision Process (MDP). The action-value function is used in many RL algorithms. It describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t|s_t = s, a_t = a, \pi] \tag{1}$$

The action-value function obeys a fundamental recursion known as the Bellman equation:

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{2}$$

We generally use the Bellman equation as an iterative update to estimate the action-value function:

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a')|s,a] \tag{3}$$

As the number of iterations approaches to infinity, value iteration algorithms are guaranteed to converge to the optimal action-value functions, $Q_i \longrightarrow Q^*$. However, it is impractical to estimate the optimal Q-value of every single state-action pair without any generalization because of the high complexity of state-action space. One of the core ideas to alleviate the computational challenge is to represent the Q-value using a function approximator such as a neural network, $Q(s,a) = Q(s,a;\theta)$, although some RL algorithms (e.g.,Q-learning) appear to be highly unstable when being combined with non-linear function approximators [6].

2.2 Deep Q-Networks

Deep Q-Networks (DQN) uses two main innovations in order to alleviate the unstability of learning when combining traditional RL algorithms with a deep convolutional neural network [2]. One key innovation is that DQN uses the experience replay mechanism. At each time-step t, the agent stores the transition tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ into a memory buffer $D = \{e_1, e_2, \ldots, e_t\}$ and then samples transitions uniformly for training. Another key innovation behind the success of DQN is the use of two separate Q-networks $Q(s,a;\theta)$ and $Q(s,a;\theta^-)$ with current parameters θ and old parameters θ^- respectively. At every updating iteration i, the current parameters θ are updated so as to minimize the mean-squared Bellman error with respect to old parameters θ^-, by optimizing the following loss function:

$$L_i(\theta_i) = \mathbb{E}\Big[\Big(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\Big)^2\Big] \tag{4}$$

Differentiating the loss function with respect to the current parameters, we arrive at the following gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \Big(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\Big) \nabla_{\theta_i} Q(s,a;\theta) \tag{5}$$

Then, the parameters of the network are adjusted in the direction of the gradient descent of the loss function:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta_i} L_i(\theta_i) \tag{6}$$

DQN is off-policy, that is to say, it estimates the optimal Q-values by the greedy strategy $a = \max_a Q(s,a;\theta)$, while selecting actions by an ε-greedy strategy with the purpose of ensuring the balance between exploration and exploitation. In brief, DQN follows the greedy action with probability $1 - \varepsilon$ and selects a random action with probability ε.

3 Model Architecture and Algorithm

3.1 Details of Deep CNN Architecture

We use a deep convolutional neural network architecture in which there is a separate output unit for predicting Q-values of discrete actions. The main advantage of this architecture is its ability to compute Q-values for all possible actions in a given state representation with only a single forward pass along the network. The exact architecture used for most deep reinforcement learning tasks, demonstrated in Fig. 1, is as follows. The model is similar to the DQN's architecture except that the full-connected layer is followed by a dropout operation [7] in order to handle the problem of over-fitting. The input to our network is a $84 \times 84 \times 4$ image produced by the preprocessing procedure [2]. The first convolution layer convolves 32 filters of 8×8 with stride 4 with the input image and followed by a rectifier nonlinearity (ReLU). The second hidden layer convolves 64 filters of 4×4 with stride 2, again applies a rectifier nonlinearity. Then the final convolution layer of our network convolves 64 filters of 3×3 with stride 1 followed by a rectifier. This is followed by a fully-connected hidden layer consisting of 512 rectifier units and then a dropout operation. Finally, the output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied from 4 to 18 on the Atari 2600 games.

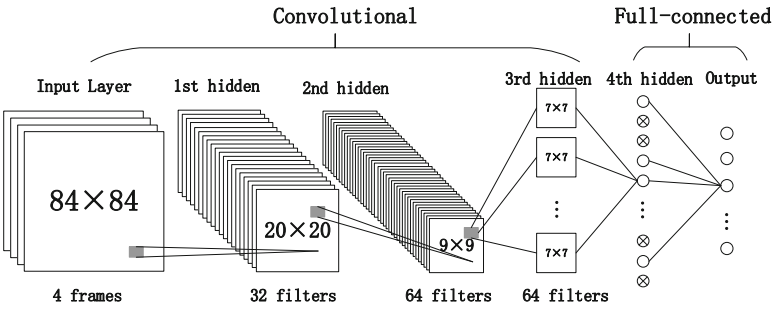

Fig. 1. The exact architecture of the deep neural network.

3.2 Prioritized Sampling

As in DQN, uniform sampling has some limitations. This is because this method does not make fully use of some valuable transitions from which we can learn the most, and always overwrites with recent transitions owing to the finite memory size. To alleviate this problem, here we use our prioritized sampling method instead of uniform sampling.

Generally, we improve the efficiency of sampling by making the transitions with positive rewards or high magnitude of TD errors to be sampled more frequently. On the one hand, the transitions that provide positive rewards are

extremely rare at the primary stage of learning. However, these transitions are more valuable and informative than others for agents to learn from. So in our algorithm, we use two separate memory buffers to improve the utilization of transitions with positive rewards. The higher priority buffer D_1 is used to store transitions with positive rewards, and naturally the lower priority buffer D_2 stores transitions with non-positive rewards. Then we use a similar stratified sampling method to sample transitions from the higher priority buffer with probability ρ and from the other with probability $1 - \rho$.

On the other hand, some fraction of transitions are never sampled before they drop out of the buffers. To alleviate this problem, we add a term v_t for tracking the sampled times in the transition tuple $e_t = (s_t, a_t, r_t, v_t, s_{t+1})$. We can assume that the frequently replayed transitions will have low TD errors because of more opportunities to modify the Q-values so as to approximate the target action-value functions. Naturally, we prefer to sample the transitions that have not been sampled for a while, because they have relatively larger TD errors. So our innovation is to use a prioritized sampling based on the sampled times of each transition for ensuring every sample is replayed from time to time. We define the priority of transition j as:

$$p_j = \frac{1}{(v_j + 1)} \tag{7}$$

As we can see from this definition, the priority of transition j monotonously decreases with the increase of sampled times.

However, greedy sampling solely based on the priorities of transitions may make the training data lack of diversity. Specifically, to avoid expensive sweeps over the entire memory buffers, greedy sampling is prone to sample the transitions with higher priorities, meaning that a transition that has a lower priority after a replay may not be sampled anymore. One consequence is that the TD errors used for updating Q-values shrink slowly, especially when using the deep neural network as a function approximator. It is necessary to propose a method that can take full advantage of the transitions' priorities and ensure the diversity of sampling at the same time.

So we introduce a stochastic sampling method that falls in between pure greedy sampling by priority and sampling uniformly. We make a guarantee that the probability of being sampled to be monotonically increased in a transition's priority, while ensuring a non-zero probability of being sampled even for a transition with the lowest priority. Specifically, we define the probability of sampling transition j as:

$$P(j) = \frac{p_j^\alpha}{\sum_{i=1}^{i=size(D_1)} p_i^\alpha} \tag{8}$$

where $p_j > 0$ is the priority of transition j. The exponent α determines the degree of prioritization when sampling transitions. It is obvious that setting $\alpha = 0$ corresponds to the uniform sampling, while $\alpha = 1$ corresponds to the pure greedy sampling case.

We combine our prioritized sampling method with a deep reinforcement learning agent known as DQN. Our main modification is to replace the uniform sampling used by DQN with our prioritized sampling method. The full algorithm is presented in the next section.

Algorithm 1. deep Q-learning with prioritized sampling

Randomly initialize Q-Network Q with weights θ and soft target Q-Network $\hat{Q}$ with weights $\theta^- \leftarrow \theta$; memory buffers D_1 and D_2 to capacity N; mini-batch M, $p_1 = 1$.
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 $a_t = \arg\max_a Q(\Phi(s_t), a; \theta)$.
    Execute action a_t and observe reward r_t and new image x_{t+1}.
    Set $s_{t+1} = \{s_t, a_t, x_{t+1}\}$, $v_t = 0$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$.
    if $r_t > 0$ **then**
      Store transition $(\Phi_t, a_t, r_t, v_t, \Phi_{t+1})$ in D_1.
    else
      Store transition $(\Phi_t, a_t, r_t, v_t, \Phi_{t+1})$ in D_2.
    for $m = 1, M$ **do**
      if random() $< \rho$ **then**
        Sample a transition $(\Phi_j, a_j, r_j, v_j, \Phi_{j+1})$ from D_1 according to a probability distribution $P(j) = (p_j)^\alpha / \sum_i (p_i)^\alpha$.
      else
        Sample a transition $(\Phi_j, a_j, r_j, v_j, \Phi_{j+1})$ from D_2 according to a probability distribution $P(j) = (p_j)^\alpha / \sum_i (p_i)^\alpha$.
      Update replayed times: $v_j = v_j + 1$.
      Update transition priority: $p_j = 1/(v_j + 1)$.
    end for
    Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(\Phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform gradient descent step on the loss $L(\theta) = (y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to the network parameters θ.
    Update the target networks: $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$.
  end for
end for

3.3 Algorithm

Directly implementing Q-learning with a deep neural network proved to be unstable in many environments. However, such non-linear function approximators appear to be necessary to learn more abstract and valuable feature representation when confronting with large state space. Therefore, we need to make two improvements to ensure the stability of our algorithm.

As in DQN, we firstly use the experience replay mechanism to address the problem of highly correlation between samples. The transitions in the form of $(s_t, a_t, r_t, v_t, s_{t+1})$ are stored into different buffers according to the magnitude of r_t. Parameters of the network are updated by performing stochastic gradient descent using a mini-batch of transitions obtained by prioritized sampling from the buffers.

The second modification aiming at ensuring the stability of our algorithm is to use a separate target network $Q(s, a; \theta^-)$ to generate the target Q-values: $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$. We use a soft target update, rather than directly copying the weights of the current network to the target network. Instead, the weights of these target networks are then updated by slowly tracking the current network: $\theta^- \longleftarrow \tau\theta + (1 - \tau)\theta^-$ with $\tau \ll 1$. Generating θ^- in this way makes the target Q-values change slowly, greatly improving the stability of learning the optimal action-value functions. The full algorithm for training the network is presented in Algorithm 1.

4 Experiments

4.1 Experimental Set up

We perform an evaluation of our proposed PS-DQN agent by conducting experiments on four Atari 2600 games using the Arcade Learning Environment [8] (ALE). ALE provides a challenging and diverse set of RL problems where an agent must learn to play the games directly from the high-dimensional sensory video inputs. In our experiments, all hyper-parameters are identical to DQN unless stated differently.

Firstly, the "soft" factor τ is set to be 0.05 for having the target Q-network slowly track the current network. The gradients are clipped to fall within [-5,5] to guarantee the stability of learning. In addition, the parameter ρ starts at 0.5 and decays linearly to 0.25 over the first million frames because smarter agent will get more transitions with positive rewards through incremental learning. For the hyper-parameter α that are utilized to ensure the diversity of transitions, we did a coarse grid search (evaluated on the game of Breakout), and found that the setting $\alpha = 0.6$ appears best in our algorithm. On each game, the network is trained on a single GPU for 50 million frames consuming one week and utilized two memory buffers with the capacity of one million most recent frames.

To summarize, our experiments only use a minimal prior knowledge consisting of the input sensory images, the game-specific scores and a single set of hyper-parameters across all games, resulting in an artificial agent with the capability of learning to being expert in a diverse of challenging tasks.

4.2 Main Evaluation

We select the following four games for evaluation: Breakout, Boxing, Pong and Space_invaders as tested problems. The deep Q-Network described in [2] is used

as a baseline. A random number of frames were skipped by repeatedly taking the null or do nothing action before giving control to the agent for ensuring variation in the initial conditions. The learned policies are then evaluated after every 250000 steps (an epoch) based on the average reward per episode obtained by running an ϵ-greedy policy with $\epsilon = 0.05$ for 125000 steps.

In RL, we usually set an evaluation metric which is the total reward the agent collects in an episode. Naturally, our first metric is the best average reward per episode of 50 epochs for the two agents. The comparisons of training processes of two agents on the four Atari games are depicted in Fig. 2.

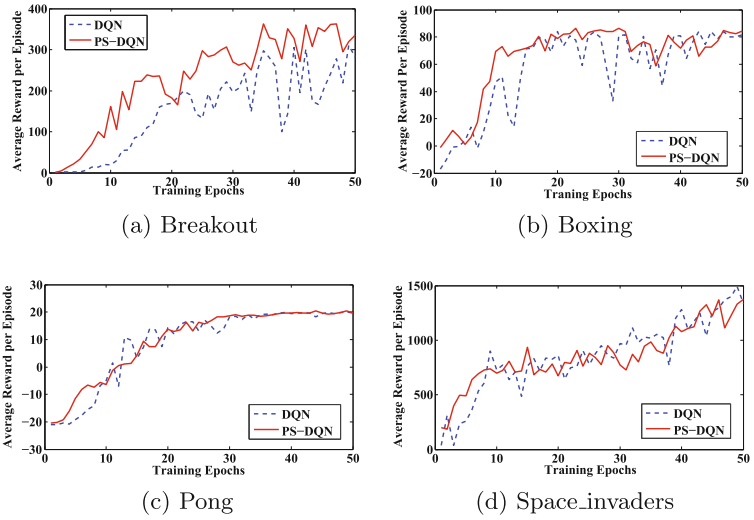

Fig. 2. The average reward per episode for the two agents on four Atari 2600 games as a function of the number of training epochs.

We find that adding prioritized sampling to DQN gives rise to a significant improvement on four games embodied in higher average scores especially at the early stage of training in most of the games. This improvement can be ascribed to the increase of utilization rate of valuable and informative transitions with positive rewards. Furthermore, we find that training agents by PS-DQN are more stable in all games with the exception of Space_invaders. This behavior caused by our prioritized sampling which makes every transition be sampled with a certain probability, rather than biasing toward out-of-date transitions which have been sampled hundreds of times. However, the average reward per episode metric tends to be noisy because small changes to the weights of a policy can lead to large changes in the distribution of states the policy visits [1]. So we used a more stable metric which is the average maximum predicted action-value function. According to the results depicted in Fig. 3, we make a conclusion that our PS-DQN algorithm leads to a faster convergence speed than DQN. On

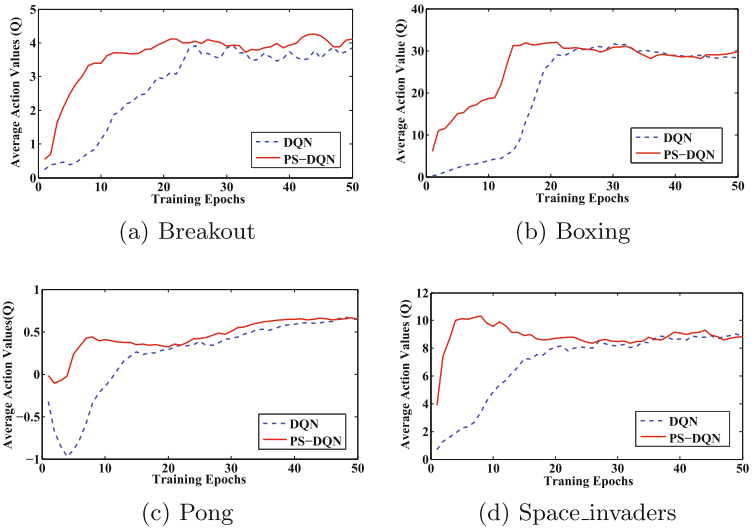

Fig. 3. The average maximum predicted Q-value per episode for the two agents on four Atari 2600 games as a function of the number of training epochs.

Breakout and Boxing, negative reward doesn't exist in the game and positive reward is rare at the early stage of training. So with our sampling mechanism, Q-values increase smoothly until the network converges. On pong, negative rewards appear frequently during the early stage of playing. So the Q-values curve which represents the learning process of DQN has a low peak as depicted in Fig. 3(c). This fluctuation of the Q-value function is adverse to the learning of the agent. Fortunately, our preference to transitions with positive rewards avoids a local minimum of average Q-values, resulting in a performance boost on the stability of learning. However, we do not lack the positive rewards at the beginning of the agent's learning on Space_invaders. Our sampling mechanism inevitably leads to an overuse of the samples with positive-reward. So it is not difficult to explain the overestimation of Q-values at the beginning of training as depicted in Fig. 3(d).

Nevertheless, our PS-DQN algorithm also has some limitations. We can see that there is almost no improvement in the curve of average reward of Space_invaders reflecting in lower average rewards for some epochs and the instability of the training process induced by overusing of the transitions with positive-rewards. The experimental results indicate that our PS-DQN agent is considerably appropriate for these sensing and controlling tasks which could generate a mass of zero rewards except for some scattered positive rewards at the primary stage of learning, such as Breakout and Boxing.

5 Conclusion and Future Work

We presented a novel algorithm, called PS-DQN, by combining the deep Q-network with a prioritized sampling strategy. According to the analysis of

our experimental results on four Atari 2600 games, we arrive at a conclusion that using prioritized sampling may lead to a faster and more stable learning process, and a better performance of scoring on some tested games. These preliminary results might provide empirical clues for further research, in particular developing an automatic way to adapt the hyper-parameter ρ on-line based on the distribution of transitions distinguished by the magnitude of rewards.

Acknowledgements. This work was funded by National Natural Science Foundation (61272005, 61303108, 61373094, 61502323, 61472262). We would also like to thank he reviewers for their helpful comments. Natural Science Foundation of Jiangsu (BK2012616), High School Natural Foundation of Jiangsu (13KJB520020), Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University (93K172014K04), Suzhou Industrial application of basic research program part (SYG201308, SYG201422).

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Playing atari with deep reinforcement learning. In: Deep Learning Workshop of the 27th Advances in Neural Information Processing Systems, NIPS, Lake Tahoe (2013)
2. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
3. Silver, D., Huang, A., Maddison, C.J., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
5. Lin, L.J.: Reinforcement learning for robots using neural networks. Technical report, DTIC Document (1993)
6. Tsitsiklis, J.N., Van, R.B.: An analysis of temporal-difference learning with function approximation. IEEE Trans. Autom. Control **42**(5), 674–690 (1997)
7. Hinton, G.E., Srivastava, N., Krizhevsky, A., et al.: Improving neural networks by preventing co-adaptation of feature detectors. Comput. Sci. **3**(4), 212–223 (2012)
8. Bellemare, M.G., Naddaf, Y., Veness, J., et al.: The arcade learning environment: an evaluation platform for general agents. J. Artif. Intell. Res. **47**(1), 253–279 (2012)

http://www.springer.com/978-3-319-46686-6

Neural Information Processing
23rd International Conference, ICONIP 2016, Kyoto, Japan, October 16–21, 2016, Proceedings, Part I
Akira, H.; Seiichi, O.; Doya, K.; Kazushi, I.; Minho, L.; Derong, L. (Eds.)
2016, XIX, 639 p. 250 illus., Softcover
ISBN: 978-3-319-46686-6