Processing Element Architecture Design for Deep Reinforcement Learning with Flexible Block Floating Point Exploiting Signal Statistics

Juyn-Da Su and Pei-Yun Tsai
Department of Electrical Engineering,
National Central University, Taiwan

Abstract—**Deep reinforcement learning is a technique that allows the agent to have evolving learning capability for unknown environments and thus has the potential to surpass human expertise. The hardware architecture for DRL supporting on-line Q-learning and on-line training is presented in this paper. Two processing element (PE) arrays are used for handling evaluation network and target network respectively. Through configuration of two modes for PE operations, all required forward and backward computations can be accomplished and the number of processing cycles can be derived. Due to the precision required for on-line Q-learning and training, we propose flexible block floating-point (FBFP) to reduce the overhead of floating-point adders. The FBFP exploits different signal statistics during the learning process. Furthermore, the respective block exponents of gradients are adjusted following the variation of temporal-difference (TD) error to reserve resolution. From the simulation results, the FBFP multiplier-and-accumulator (MAC) can reduce 15.8% of complexity compared to FP MAC while good learning performance can be maintained.**

Index Terms—**Block floating-point, deep Q network, reinforcement learning, architecture design.**

I. Introduction

Due to the successful experience of AlphaGo Zero, which learns the game of Go from scratch without human knowledge based on the deep reinforcement learning (DRL) algorithm [1], researchers tend to employ DRL in various application domains to exploit the possibilities that it brings. Studies on DRL in mobile robot navigation are popular in order to combat the dynamically changing environments [2][3]. In addition, DRL can also aid in deployment of internet of things (IoT) that needs decision process for resource allocation or scheduling [4][5]. Thus, DRL has been shown a potential technique to incorporate evolving learning capability for signal processing and system design.

Although the rapid growth of massive computations enabled by parallel processing of GPU benefits the development of DRL, implementation and realization of DRL in mobile or edge devices must take performance and power consumption into consideration. Some dedicated hardware accelerators have been presented in the literature. A Q-learning processor is designed on FPGA in [6], which uses fixed point for a 8×4 Q-matrix. An AI processor in 65nm CMOS technology with processing elements for tree search and reinforcement learning is reported in [7] for mobile robot navigation, which consumes 1.1mW at 0.55V. A planetary navigation robot is implemented on FPGA in [8], and the Q-learning algorithm with multi-layer perceptron containing 11 neurons for simple environments and 25 neurons for complex environments is adopted. Both fixed point and floating point (FP) are considered for datapath. In [9], a general framework for DRL utilizing stochastic computing is described, which includes off-line DNN construction and on-line deep Q-learning, providing the hardware synthesis results in CMOS technology. In [10], a neuromorphic accelerator using stochastic neural network plus Q learning with 6-bit multiplier-and-accumulator (MAC) units is fabricated in 55nm CMOS technology. We can see that the hardware accelerators have different coverages, from reinforcement Q learning to DRL and from off-line training to on-line training. Consequently, the hardware costs and performances are quite different.

Datapath quantization is an important task to tradeoff performance and complexity in hardware implementation and has been investigated widely in the designs of deep neural network for supervised learning. For inference applications, fixed-point representation is frequently used to save power and complexity with slightly sacrificed accuracy [11]. On the other hand, when both training and inference are taken into consideration, 32-bit floating point and customized 16-bit floating point are used in [12]. As mentioned in [13], the dynamic range provided by fixed point is usually insufficient for deep neural network training convergence and they propose hybrid block floating point for hardware supporting training for general applications. All dot products are performed in block floating-point and the remaining blocks are operated in floating point format so as to gain some advantage of the fixed-point arithmetic design for dot product.

In this paper, architecture is first developed to support both on-line DRL training and Q-learning. In order to save the complexity for arithmetic units of a general DRL accelerator and to ensure the training convergence for various applications, we exploit the data statistics and propose to use FBFP in the PEs that deal with both weight update and Q-value computation. The simulation results from the cycle-accurate bit-true model show that the FBFP datapath works well and the agent still achieves good performance compared to the floating-point design. In addition, from the synthesis results, the 15.8% area complexity can be saved if the FBFP adders are adopted in the MAC of the PE compared to the FP MAC at 400MHz operating frequency in 40nm CMOS technology.

In the following, deep reinforcement learning with prioritized experience replay is introduced in Sec. II. Sec. III describes the architecture and scheduling for on-line training and Q-learning. Sec. IV discusses the performance and complexity comparison of FBFP and FP. Conclusion is given in Sec. V.

II. Deep Q Network with Prioritized Experience Replay

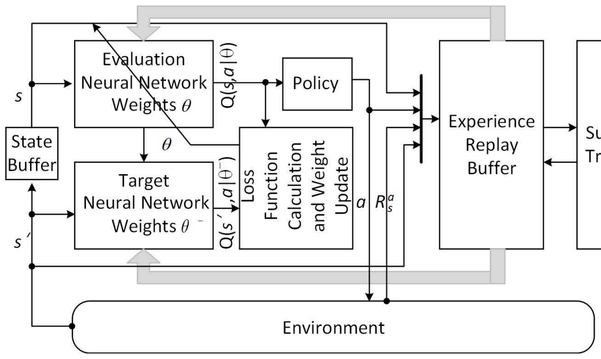
Evaluation Neural Network Weights θ
Policy
Experience Replay Buffer
Sum Tree
State Buffer
Target Neural Network Weights θ⁻
Loss Function Calculation and Weight Update
Q(s,a|θ)
Q(s′,a|θ⁻)
s
s′
θ
a
R_s^a
Environment

Fig. 1 Flow of DQN.

Reinforcement learning, one of the machine learning techniques, considers the task of an agent interacting with the environment. Usually, it is modelled by the Markov decision process described by state S_t, action A_t, and reward R_t at each step t. The agent aims to find a policy π to optimize the long-term return defined by

$$Q_\pi(s,a) = E\{G_t|S_t = s, A_t = a\}, \tag{1}$$

where

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2}$$

with the discount factor $0 \le \gamma \le 1$.

Deep reinforcement learning, which uses high dimensional sensor data and neural network to learn the control policy, was presented in [14]. Two networks are employed to approximate the action-value function, $Q(s,a|\theta)$ from the evaluation network and $Q(s',a|\theta^-)$ from the target network as shown in Fig. 1, where s is the current state and s' is the next state. For each state s, the agent determines the action a derived from $Q(s,a|\theta)$, $\forall a$, and the ε-greedy strategy. The agent then observes the reward R_s^a fedback by the environment. The transition information (s,a,R_s^a,s') is saved in the experience replay buffer. Given mini-batch size B, B transitions are sampled from the replay buffer according to their priority [15]. Define

$$y_s^a = \begin{cases} R_s^a & \text{terminal } s' \\ R_s^a + \gamma \max\limits_{a'} Q(s',a|\theta^-) & non-terminal\ s' \end{cases}. \tag{3}$$

The temporal difference (TD) error is computed by

$$\delta = y_s^a - Q(s,a|\theta). \tag{4}$$

The loss function is defined as,

$$\mathcal{L}(\theta) = \frac{1}{2}\delta^2. \tag{5}$$

The weights θ of the evaluation network are updated by the gradient descent to minimize the loss. The weight θ^- of the target network is set to the same weight as the evaluation network every C steps. These steps of on-line training and Q-learning repeat until the state s' is a terminal to end an episode.

III. Processing Element Architecture for Deep Q Network

We design a generalized hardware accelerator to support the deep Q network operations with on-line training and Q-learning.

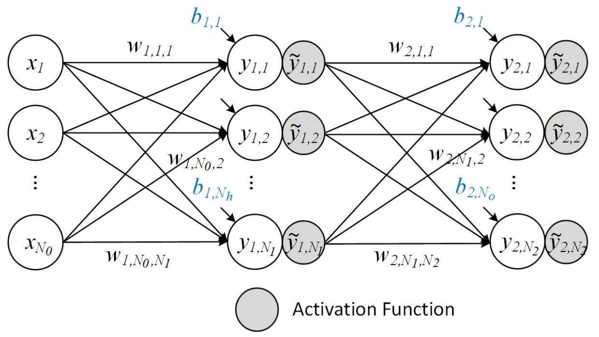

Fig. 2 Neural network for DQN

The neural network for DQN is described in Fig. 2, where N_l denotes the number of neurons in the lth layer. To support on-line Q-learning and training, the accelerator must be capable of performing the following operations.

- On-line Q-learning

Forward propagation in neural network is required for on-line Q-learning that uses $Q(s,a|\theta)$ and $Q(s',a|\theta^-)$ as described in (1), which takes the form of

$$y_{l,n} = \sum_{m=1}^{N_{l-1}} w_{l,m,n}\tilde{y}_{l-1,m} + b_{l,n}, \tag{6}$$

and

$$\tilde{y}_{l,n} = \phi(y_{l,n}) \tag{7}$$

where l is the layer index, $\phi(\cdot)$ is the activation function, and $\tilde{y}_{0,m} = x_m$. Then, $Q(s,a|\theta) = \max_n \tilde{y}_{2,n}$ given state $s = [x_0\ x_1 \dots x_{N_0}]$.

- On-line training

Backward propagation in neural network is also needed for on-line training. The update equation can be derived by

$$\begin{aligned}\theta' &= \theta + \alpha(y_s^a - Q(s,a|\theta))\nabla_\theta Q(s,a|\theta) \\ &= \theta + \alpha\Delta\theta = \theta + \alpha\delta\nabla_\theta Q(s,a|\theta),\end{aligned} \tag{8}$$

where α is the learning rate. If the output layer has no activation function,

$$\Delta b_{2,n} = \frac{\partial\mathcal{L}(\theta)}{\partial b_{2,n}} = \delta, \tag{9}$$

$$\Delta w_{2,m,n} = \frac{\partial\mathcal{L}(\theta)}{\partial w_{2,m,n}} = \delta\tilde{y}_{1,m} = \Delta b_{2,n}\tilde{y}_{1,m}. \tag{10}$$

For the hidden layer,

$$\begin{aligned}\Delta b_{1,m} &= \frac{\partial\mathcal{L}(\theta)}{\partial b_{1,m}} = \delta w_{2,m,n}\dot{\phi}(y_{1,m}) \\ &= \dot{\phi}(y_{1,m})\Delta b_{2,n} w_{2,m,n},\end{aligned} \tag{11}$$

$$\Delta w_{1,k,m} = \frac{\partial\mathcal{L}(\theta)}{\partial w_{1,k,m}} = \Delta b_{1,m} x_k, \tag{12}$$

where $\dot{\phi}(\cdot)$ is the first derivative of the activation function. Note that TD error δ exists in $\Delta b_{l,n}$ and $\Delta w_{l,m,n}$ in Eqs. (9)-(12), the gradient of the loss function with respect to $b_{l,n}$ and $w_{l,m,n}$ for $l = 1,2$. It means that the magnitude of the TD error influences the magnitude of the gradient for parameter update.

On-line Q-learning needs the results of $Q(s,a|\theta)$ and $Q(s',a|\theta^-)$. Thus, two sets of processing element (PE) arrays are used, one for $Q(s,a|\theta)$ and the other for $Q(s',a|\theta^-)$. For

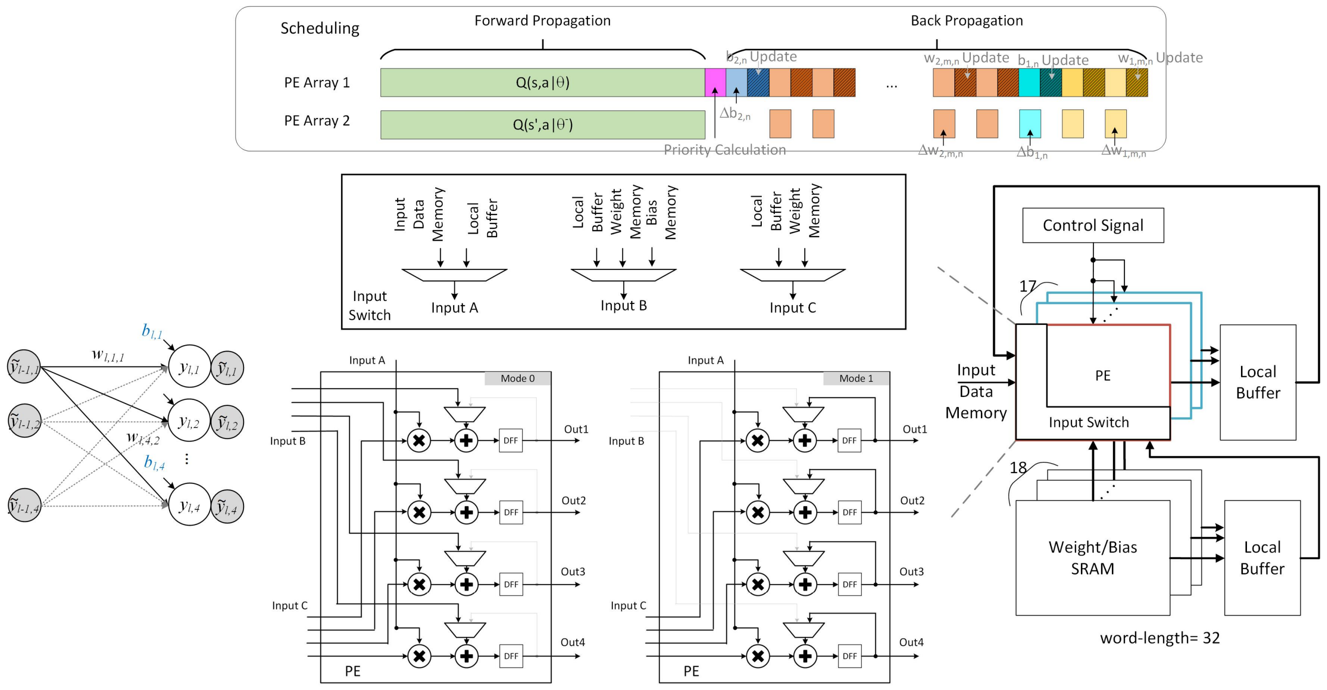

Fig. 3 Architecture of PE Array

each PE array, K PEs are employed each having four multipliers and accumulators, as shown in Fig. 3. The input reuse architecture is adopted for input A. In order to provide configurability, multiplexers are inserted to change the signal flows. For the forward propagation (FP) of on-line Q learning, initially, the PE is configured in Mode 0. Variable $\tilde{y}_{l-1,m}$ and $b_{l,n}$ are fed from input A and input B, respectively. Input C then sends the weight $w_{l,m,n}$. Next, the PE is switched into Mode 1, which computes the partial sum of Eq. (6) in each clock cycle t.

$$\Lambda_{m,n}^{(t+1)} = w_{l,m,n}\tilde{y}_{l-1,m} + \Lambda_{m,n}^{(t)}. \tag{13}$$

with $\tilde{y}_{l-1,m}$ from input A and $w_{l,m,n}$ from input B. It takes $N_{l-1}N_l/(4K)$ clock cycles to complete the partial sum calculation. Note that $y_{1,n}$ will pass through the activation function and the derivative of activation function. Both outputs are stored in the local buffer for subsequent use. In addition, the partial sums must be added together to generate the final Q value, $Q(s,a|\theta)$ or $Q(s',a|\theta^{-})$. The PE is then set to Mode 1 again but performs tree addition with input A equal to 1. It takes $\lceil log_2(S) \rceil$ clock cycles if there are S partial sums.

To support the back propagation (BP) for computing the gradient $\Delta w_{2,m,n}$ described in Eq. (10), the PEs can be configured to Mode 0, which performs multiplication only with input B equal to zero. There are N_1 multiplications. If two PE arrays are utilized concurrently, $\lceil N_1/(8K) \rceil$ clock cycles are required. As to Eq. (11) about the gradient $\Delta b_{1,m}$, Mode 0 is still used for multiplication of $\Delta b_{2,n}$ and $w_{2,m,n}$ to generate the intermediate result Λ_m. To complete the calculation of $\Delta b_{1,m}$ in Eq. (11), $\dot{\phi}(y_{1,m})$ is fetched from local buffer and the multiplication of Λ_m and $\dot{\phi}(y_{1,m})$ is performed. Note that each PE supports only vector multiplication by a scalar. Consequently, only one among four outputs is valid and it takes

Table I Input settings and processing cycles of the respective step.

	Input A	Input B	Input C	Clock Cycles
FP Initialization (Mode 0)	$\tilde{y}_{l-1,m}$	$b_{l,n}$	$w_{l,m,n}$	$\lceil N_l/(4K) \rceil$
FP Partial Sum (Mode 1)	$\tilde{y}_{l-1,m}$	-	$w_{l,m,n}$	$N_{l-1}N_l/(4K)$
Tree Addition (Mode 1)	1	$\Lambda_{m_1,n_1}^{(t)}$	$\Lambda_{m_2,n_2}^{(t)}$	$\lceil log_2(S) \rceil$
BP Mul. (10) (Mode 0)	$\Delta b_{2,n}$	0	$\tilde{y}_{1,m}$	$\lceil N_1/(8K) \rceil$
BP Mul. (11) (Mode 0)	$\Delta b_{2,n}$	0	$w_{2,m,n}$	$\lceil N_1/(8K) \rceil$
BP Mul. (11) (Mode 0)	Λ_m	0	$\dot{\phi}(y_{1,m})$	$\lceil 4N_1/(8K) \rceil$
BP Mul. (12) (Mode 0)	$x_{0,k}$	0	$\Delta b_{1,m}$	$N_0N_1/(8K)$
L1 Parameter Update (Mode 0)	α	θ	$\Delta\theta$	$N_0N_1/(8K)$, $\lceil N_1/(8K) \rceil$
L2 Parameter Update (Mode 0)	α	θ	$\Delta\theta$	$N_1/(8K)$, $\lceil 1/(8K) \rceil$

$\lceil 4N_1/(8K) \rceil$ clock cycles. The gradient $\Delta w_{1,k,m}$ can be obtained by multiplication of $\Delta b_{1,m}$ and x_k. Therefore, we can use Mode 0 with x_k from input A and $N_0N_1/(8K)$ clock cycles are required. The parameter update needs $N_1N_2/(8K)$ clock cycles for weights and $\lceil N_1/(8K) \rceil$ clock cycles for bias in layer 1. The PEs remain in Mode 0. The learning rate α is assigned to input A, the parameter θ is sent from input B. The controls and settings for each step are summarized in Table I.

IV. Datapath with Flexible Block Floating Point

To verify the datapath precision requirements of on-line Q-learning and training, a maze problem of size 12×12 is employed for testing. If the robot arrives at the destination, reward of +1 will be obtained. On the other hand, if the robot falls in a trap, reward of -1 will be received. Both the destination and the trap are terminal states which end the episode. An example is given in Fig. 4. The black area denotes the blockage. The numbers of neurons, N_0, N_1, and N_2 in each layer, are set to 144, 72, and 4, respectively. Define win rate $\mathcal{D}_i$ of the ith epoch as

$$\mathcal{D}_{\mathrm{i}} = \frac{1}{E}\sum_{j=0}^{E-1} S(i-j), \tag{14}$$

where $E = 200$ and $S(i) = 1$ if the robot can arrive at the destination in the ith episode successfully. Otherwise $S(i) = 0$. Fig. 5 shows the simulation results of the epoch win rate versus mantissa wordlength (WL) of the multipliers and the adders in PE arrays given the cycle-accurate model described in Fig. 3. It is clear that if the mantissa wordlength is not sufficient, the DRL accelerator may not converge. The IEEE-754 floating-point (FP) representation uses 23 bits for mantissa and thus almost floating-point precision is required for the datapath of the DRL accelerator.

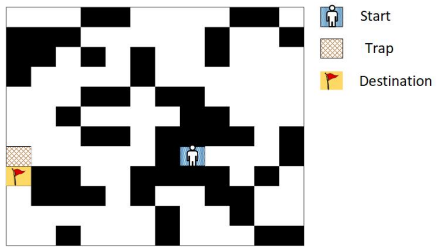

Fig. 4 Maze of size 12×12.

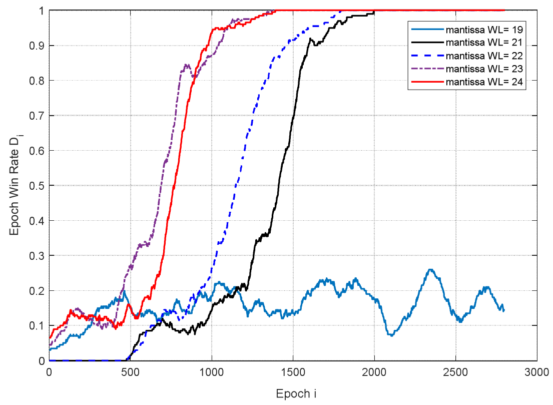

Fig. 5 Datapath precision requirement.

The floating-point representation incurs higher complexity of the adders and multipliers compared to the fixed-point representation. In order to further reduce the complexity of the processing elements, the variations of the dynamic ranges of different variables are observed. Fig. 6 shows the distributions of the exponents of the gradient $\Delta b_{2,n}$ and $\Delta w_{1,k,m}$ at the early and late stages during the learning process. The wider dynamic range is required at the early stage than at the late stage. In addition, when the network gets converged, the TD error becomes small and so do gradients. If the maximal exponent of the TD error is denoted as $\mathcal{E}_{\delta,\max}$, then from (9) and (10), the maximal exponent of $\Delta b_{2,n}$ and $\Delta w_{2,m,n}$ can be expressed as

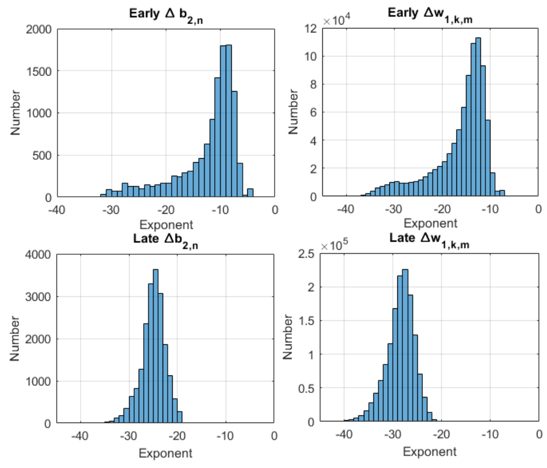

Fig. 6 Distribution of exponents from gradient at early and late stages.

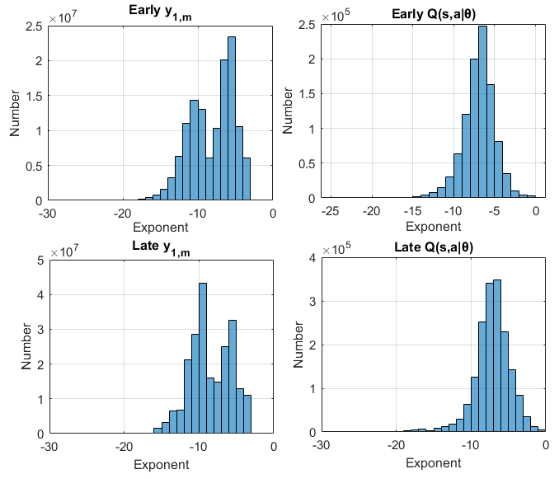

Fig. 7 Distribution of exponents from the hidden layer and Q values at early and late stages.

$$\mathcal{E}_{\Delta \mathrm{b}_2,\max} = \mathcal{E}_{\delta,\max} \tag{15}$$

$$\mathcal{E}_{\Delta \mathrm{w}_2,\max} = \mathcal{E}_{\delta,\max} + \mathcal{E}_{\tilde{y}_1,\max} - \mathcal{E}_{\mathrm{offset}} \tag{16}$$

where $\mathcal{E}_{\tilde{y}_1,\max}$ is the maximal exponent of layer 1 output $\tilde{y}_{1,m}$ and $\mathcal{E}_{\mathrm{offset}} = 127$. From (11) and (12), the maximal exponent of $\Delta b_{1,m}$ then becomes

$$\mathcal{E}_{\Delta \mathrm{b}_1,\max} \leq \mathcal{E}_{\delta,\max} + \mathcal{E}_{\mathrm{w}_2,\max} + \mathcal{E}_{\phi,\max} - \mathcal{E}_{\mathrm{offset}} \tag{17}$$

Note that $\mathcal{E}_{\phi,\max} = -2$ for sigmoid function and $\mathcal{E}_{\phi,\max} = 0$ for Relu function and hyperbolic tangent. Besides,

$$\mathcal{E}_{\Delta \mathrm{w}_1,\max} = \mathcal{E}_{\Delta \mathrm{b}_1,\max} + \mathcal{E}_{\mathrm{x}_0,\max} - \mathcal{E}_{\mathrm{offset}}. \tag{18}$$

Therefore, the maximal exponent of these gradients can be estimated according to $\mathcal{E}_{\delta,\max}$. Fig. 7 shows that the distribution of exponents from hidden layer outputs and Q values. It is interesting that the dynamic ranges of these variables do not vary too much at the early and late stages.

The exponents of the augend and addend must be aligned to the maximum of them for the floating-point addition. Consequently, for those calculations that use Mode-1 accumulation of the PE in Table I, the precise floating-point

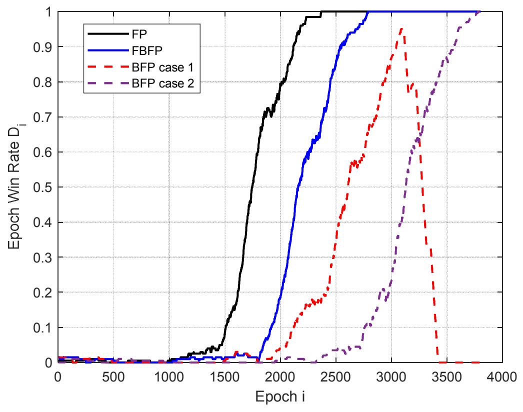

Fig. 8 Performance comparison for different settings about BFP and FP.

addition causes repeatedly realignments during the accumulation. Drumond et al. mentioned about this phenomenon in [13] and hybrid block floating-point instead of block floating-point (BFP) is used for the deep neural network training. Only the dot operations are handled by the block floating-point and the remaining operations are still performed in floating-point calculation. FP-to-BFP conversion is required and inserted in the interface of dot-operations.

Here, we propose to use flexible block floating-point design for the DRL accelerator, which needs not only on-line training but also on-line Q-learning. Note that the alignment of exponents of augend and addend and readjustment the exponent of the sum result in a large overhead of a floating-point adder compared to a fixed-point adder. From the scheduling of PE operations discussed in Sec. III, flexible block floating-point is applied to the data processing utilizing Mode-1 accumulation according to the respective data statistics without the FP to BPF conversion. Therefore, eight D flip-flops are adopted to record the block exponents used for the accumulations of the respective results: hidden layer outputs, FP partial sum, FP tree addition, layer-1 weights, layer-1 bias, layer-2 weights, layer-2 bias, and gradients. These values except the last one can be obtained from the observations of the early learning process. For example, given offset of 127, the block exponent of hidden layer outputs is set to 125 from Fig. 7.

Since TD error is gradually decreased and influenced the magnitude of gradients, thus we use floating-point to calculate TD error in Eq. (4) and obtain its true exponent. PE1, marked as red block with a floating-point adder, is reserved for TD error calculation. After the true exponent of TD error is derived, the maximum exponent of TD errors in 50 episodes is employed as the shared block exponent of gradients so as to reserve precision for gradients.

For FBFP datapath, the fixed-point adder without post alignment can be used, the same as the BFP datapath. However, BFP needs a maximum searcher for a block of data. The maximal exponent is predefined for different variable groups with FBFP. In addition, the adjustment of the block exponent of gradient variables is not derived from the variables themselves, but derived from the TD errors, which is usually decreasing gradually during the learning process. Thus, FBFP saves the complexity and latency for searching the maximal exponent of a data block.

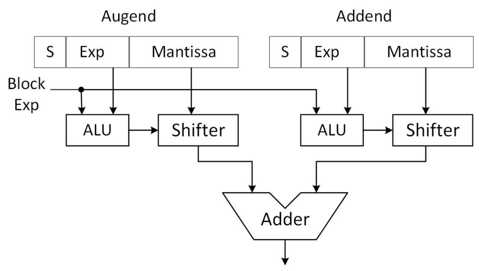

Fig. 9 Block floating-point adder.

Table II: Complexity Comparison.

Area (μm^2)		200MHz	400MHz
FP MAC [8]	Adder	1244	1610
	Multiplier	2118	3254
	MAC	3946	5441
FBFP MAC	Adder	703	963
	Multiplier	2331	3001
	MAC	3651	4582
Adder Improvement		43.4%	40.2%
MAC Improvement		7.47%	15.79%

To verify the feasibility of the flexible block floating-point, a cycle-accurate bit-true model is constructed and several settings are compared, including FP, FBFP, BFP with one shared exponent (BFP case 1) which is equivalent to fixed point, and 8 separate BFP with fixed exponent for gradients (BFP case 2). According to the operation configuration defined in Table I, the corresponding exponent will be used during the PE processing. Namely, the variation of TD error does not change the respective block exponents of gradients. The simulation result is given in Fig. 8. The accelerator using BFP with only one shared exponent for accumulation can not achieve converged learning performance. The learning performance of the accelerator using FBFP is better than that using BFP case 2 because the precision of gradients becomes worse at the late stage during the learning process in BFP case 2. Hence, it is also clear that compared to FP, FBFP can attain hardware reduction with slight performance degradation.

The complexity saving is then investigated. Implementation of DRL with FP is described in [8]. The FP and FBPF MACs are designed respectively in 40nm CMOS technology with 23-bit mantissa at the same timing constraint of 200MHz and 400Mz operating frequency. The FBFP adder is shown in Fig. 9. No post-alignment is required after the adder. The complexity comparison in 40nm CMOS technology is given in Table II. The area of a FBPF adder is only about 60% of an FP adder. The area reduction of the FBFP MAC is about 15.8% compared to the FP MAC at the timing constraint of 2.5ns.

V. Conclusion

In this paper, the PE architecture and scheduling for DRL are designed. Two PE arrays are constructed, one for evaluation network calculation and the other for target network calculation during forward propagation. During backward propagation, two

PE arrays are adopted for gradient calculation and parameter update. The PE can be configured in two modes that can support all the required operations. Because of on-line Q-learning and on-line training, the datapath requires high precision. In order to save hardware complexity, FBFP is proposed which exploits the signal statistics and the block exponent of gradients can be adjusted according to the variation of TD errors. Simulation results show the feasibility of using FBFP and the small performance degradation compared to the one using FP. However, the FBPF can achieve 15.8% hardware saving for the complexity at high operating frequency. Consequently, feasibility of the DRL accelerator with FBFP supporting on-line Q-learning and training is demonstrated.

References

[1] D. Silver et al. "Mastering the game of Go without human knowledge," Nature, 50(7676):354-359, Oct. 2017.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," IEEE Signal Processing Magazine, vol. 34, pp. 26–38, 2017.

[3] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, S. Levine, "Self supervised deep reinforcement learning with generalized computation graphs for robot navigation," 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018.

[4] J. Zhu, Y. Song, D. Jian, H. Song, "A New Deep-Q-Learning-Based Transmission Scheduling Mechanism for the Cognitive Internet of Things," IEEE Internet of Things Journal, Vol. 5, No. 4, pp. 2375-2385, Aug. 2018.

[5] Y. Wei, F. R. Yu, M. Song, Z. Han, "Joint Optimization of Caching, Computing, and Radio Resources for Fog-Enabled IoT Using Natural Actor–Critic Deep Reinforcement Learning," IEEE Internet of Things Journal, vol. 6, No. 2, pp. 2061-2073, Apr. 2019.

[6] S. Spano et al. "An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm," IEEE Access, vol. 7, pp. 186340-186351, Dec. 2019.

[7] Y. Kim et al., "A 0.55 V 1.1 mW Artificial Intelligence Processor with On-Chip PVT Compensation for Autonomous Mobile Robots," IEEE Trans. Circuits Syst. I. Reg. Papers, pp. 567-580, Aug. 2017.

[8] P. R. Gankidi et al, "FPGA Architecture for Deep Learning and its application to Planetary Robotics," in Proc. 2017 IEEE Aerospace Conf., pp. 1-9.

[9] H. Li, T. Wei, A. Ren, Q. Zhu, and Y. Wang, "Deep reinforcement learning: framework, applications, and embedded implementations," 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2017, pp. 847-854.

[10] A. Amravati, S. B. Nasir, S. Thangadurai, I. Yoon, A. Raychowdury, "A 55nm Time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots," 2018 IEEE International Solid - State Circuits Conference - (ISSCC), pp. 124-126.

[11] C. Sakr and N. Shanbhag, "An analytical method to determine minimum per-layer precision of deep neural networks," in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Apr. 2018, pp. 1090-1094.

[12] S. Shukla et al., "A Scalable Multi-TeraOPS Core for AI Training and Inference" IEEE Solid-State Circuits Letters, Early Access, 2019.

[13] M. Drumond, T. Lin, M. Jaggi, and E. Falsafi, "Training DNNs with hybrid block floating point," arXiv:1804.01526v3 [cs.LG], 2018.

[14] V. Mnih, et al. "Human level control through deep reinforcement learning." Nature, 518 (7540):529–533, 2015.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in International Conference on Learning Representations, Nov 2016.