International Journal of Machine Learning and Cybernetics (2021) 12:1031–1040
https://doi.org/10.1007/s13042-020-01218-z

ORIGINAL ARTICLE

Dynamic fusion for ensemble of deep Q-network

Patrick P. K. Chan[1] · Meng Xiao[1] · Xinran Qin[1] · Natasha Kees[1]

Received: 9 July 2020 / Accepted: 30 September 2020 / Published online: 16 November 2020

Abstract
Ensemble reinforcement learning, which combines the decisions of a set of base agents, is proposed to enhance the decision making process and speed up training time. Many studies indicate that an ensemble model may achieve better results than a single agent because of the complement of base agents, in which the error of an agent may be corrected by others. However, the fusion method is a fundamental issue in ensemble. Currently, existing studies mainly focus on static fusion which either assumes all agents have the same ability or ignores the ones with poor average performance. This assumption causes current static fusion methods to overlook base agents with poor overall performance, but excellent results in select scenarios, which results in the ability of some agents not being fully utilized. This study aims to propose a dynamic fusion method which utilizes each base agent according to its local competence on test states. The performance of a base agent on the validation set is measured in terms of the rewards achieved by the agent in next n steps. The similarity between a validation state and a new state is quantified by Euclidian distance in the latent space and the weights of each base agent are updated according to its performance on validation states and their similarity to a new state. The experimental studies confirm that the proposed dynamic fusion method outperforms its base agents and also the static fusion methods. This is the first dynamic fusion method proposed for deep reinforcement learning, which extends the study on dynamic fusion from classification to reinforcement learning.

Keywords Ensemble · Deep reinforcement learning · Dynamic fusion · Deep Q-network

1 Introduction

Deep Reinforcement Learning (DRL) [18] achieves exemplary performance in many decision-making applications, *e.g.,* Go [28], Auto-Driving [25], Robotics [18], Portfolio management [16], and Atari games [22]. DRL is a learning paradigm which aims to maximize the cumulative reward achieved by the interaction of an agent with an environment. Since an agent learns via the interactions with an environment without any supervision, one of the drawbacks of DRL is the training complexity [18, 25]. One possible solution is ensemble DRL [14, 34] which makes a decision by combining a set of base agents with reasonable performance. Many studies [5, 9, 26] suggest that an ensemble DRL model may achieve better results than a single DRL model because of the base agents' cooperation, *i.e.,* a misstep of a base agent can be corrected by the other agents. In addition, ensemble is more practical, as looking for the optimal agent is often time consuming and computationally taxing [3].

The fusion method plays an important role in ensemble as it decides how to combine the decisions of the base agents. Some studies assume all agents have the same ability by using majority voting [12, 14] and simple average [1, 34] fusion, but this assumption may not hold true in practice. Under such assumptions, these fusion methods may not perform well if the agents' performance differs considerably since a majority of poorly performing agents may dominate the decision. Some other fusion methods are biased towards the base agents with better overall performance measured on the training or evaluation sets, *e.g.,* a larger weight is assigned to the agent with better performance in weighted

✉ Meng Xiao
msunshine.xiao@gmail.com

Patrick P. K. Chan
patrickchan@scut.edu.cn

Xinran Qin
qinxinrancs@163.com

Natasha Kees
natasha.kees@yahoo.com

[1] School of Computer Science and Engineering, South China University of Technology, Guangzhou, China

average [4, 5]. However, a base agent may perform poorly on average but could have excellent performance in select cases. This valuable contribution may be ignored due to a small weight assignment. As a result, we observe that the existing static fusion methods do not take full advantage of base agents.

Our study aims to provide a fusion method that fully utilizes the ability of all base agents. Since an agent with exceptional performance in select situations should not be overlooked solely because of it's poor overall performance, the contribution of each agent should be evaluated according to it's local competence, as opposed to overall competence, in test states. Dynamic fusion assigns a weight value according to the local competence of an agent on the region where a test state located. To overcome this drawback of static fusion, some studies propose dynamic fusion methods for classification problems [6, 10, 21], however, to the best of our knowledge, no dynamic fusion method has been proposed for DRL yet. This paper extends the study of dynamic fusion from classification to RL.

One critical problem in dynamic fusion is how to calculate the local competence for a new sample. In classification, a set of samples is collected as an evaluation set to evaluate the performance of each base model, with classification accuracy commonly used as a criterion [6, 15, 35]. The similarity of the new sample to every other sample in the evaluation set is then calculated. The local competence of a classifier is calculated based on the accuracy on a validation sample and its similarity to the test sample [6]. Therefore, the local competence is high if a base model correctly classifies the evaluation samples similar to the test sample and vice versa.

In contrast to classification, the performance evaluation on an agent in RL is much more complicated, because the performance of an agent is affected by a sequence of actions as opposed to an independent decision in classification. Moreover, the decisions of an agent cannot be evaluated precisely as the data is not labeled. In our proposed model, the quality of an action decided by an agent on a state is quantified by the n-step cumulative reward, defined as the accumulative reward achieved in the next n steps according to a policy after taking an action. Another difference with classification is that validation data is usually given, and the validation set used in our model is generated randomly, aiming to represent all scenarios of an application. As a result, the performance of each base agent on the validation set is then evaluated in terms of the n-step cumulative reward. Additionally, the similarity of a given test state to each state in the validation set is defined as the Euclidean distance in the latent feature space learned by a base agent. The latent feature space is used because of its close relation to the decision, making it more meaningful than the more commonly used input space. Therefore, two states close to each other indicates that an agent should have similar performance. Finally, the weight of a base agent is adjusted by the current n-step cumulative reward and similarity measurements, and the previous weight value is considered since the consecutive actions and states are closely related and play an important role in achieving good performance.

A Deep Q-Network (DQN) is used as a base agent to demonstrate the proposed dynamic fusion method since DQN is one of the most popular DRL methods and gets exceptional results in many decision-making applications, *e.g.,* Auto-Driving [25], and Atari games [23]. An ensemble DQN with our fusion method is evaluated and compared with the existing static methods in three Atari games [23], including Breakout, SpaceInvaders and Seaquest in terms of accumulative reward and running time.

The rest of the paper is organized as follows. The background of this study is discussed in Sect. 2. In Sect. 3, the proposed dynamic fusion method is devised, while the experimental results and discussion are given in Sect. 4. The paper is concluded and future work is discussed in Sect. 5.

2 Background

The related concepts of this study, including RL, DQNs, and ensemble DRL, are introduced briefly in this section.

2.1 Reinforcement learning and deep Q-networks

After an agent takes the action a_t in state s_t at time t, the environment ξ returns the next state s_{t+1} and reward r_t. The cumulative reward R is defined as the summation of r_t for any t in an episode. An agent is defined as a policy function (*i.e.,* $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$) mapping a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$, where θ is the parameters of the agent. The aim of RL is to maximize the cumulative reward by a sequence of actions.

RL can mainly be divided into policy-based and value-based RL. Policy-based RL returns the probability of an action [17, 27], while the value of an action is estimated in value-based RL [20, 23]. The Q-Learning algorithm [32] is a popular value-based RL method. The action-value function $Q_\pi(s_t, a_t) = E_\pi[r|s_t, a_t]$ represents the expected reward obtained by taking action a_t according to π at state s_t at time t. The optimal policy $\pi_*(s_t)$ returns the action a yielding the maximum r in s among all available actions, *i.e.,* $\pi_*(s_t) = \arg\max_{a \in A} Q_\pi(s_t, a_t)$. The Q-values are updated iteratively according to the Bellman optimality equation defined as:

$$Q_\pi(s_t, a_t) = E_\pi[r + \tau \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] \tag{1}$$

where τ denotes the reward decay parameter which indicates the importance of future rewards. In traditional Q-learning, a Q-table is used to store the q value of each state-action pair.

However, this method cannot efficiently handle a complex task containing a larger number of states and actions [23, 30]. To overcome this issue, the Deep Q-Network (DQN) [23] is proposed, in which the mapping between a state and actions are estimated by a Deep Convolutional Neural Network (DCNN). It extracts the information from a state in the pixel level, and outputs the Q value for each action. DQNs usually exclude the pooling layer in order to preserve the location information. Another difference is that DQNs use a separate target Q-network and experience replay to increase the learning stability. DQNs achieve excellent performance in various applications, *e.g.,* Atari games [7, 23], autonomous driving [38] and robot manipulation [18]. However, the training complexity is one of the drawbacks in terms of the long training time and convergence problems.

2.2 Ensemble deep reinforcement learning

Research of ensemble methods mainly focus on supervised [15, 29] and unsupervised learning [36, 37] , and their results show that ensemble performs better than a single model in certain situations [21, 31]. Rather than seeking an optimal model, a number of diverse base models with reasonable ability are combined using a fusion method. Many studies [8, 24, 37] demonstrate that ensemble techniques outperform single ones experimentally, while some theoretical proofs show that ensemble techniques achieve more accurate results under particular assumptions, *e.g.,* the base decision makers are independent of each other [11]. One possible explanation on the superiority of ensemble is the complement of base decision markers [11], *i.e.,* a wrong decision from a decision marker may be corrected by others. There are only a few discussions on ensemble in DRL. All studies focus on static fusion [1, 13, 19], in which the way of combining base agents is fixed. Some basic summation functions, *e.g.,* average and voting, are evaluated by assuming the ability of base agents is similar [1]. The performance of base agents is considered in weighted average [4] and rank voting [33] in order to make a bias towards the agents with better performance. However, these methods may not fully utilize the knowledge of base models since the decisions from base agents with poor performance on average are usually ignored. However, a base agent with poor average performance may be an expert in some states [10, 21].

3 Proposed dynamic fusion of reinforcement learning

A base agent with poor overall performance may perform well in select states, but a static fusion method may not fully utilize this knowledge. This section introduces dynamic fusion which assigns a weight value of a base agent according to its local competence for a test state. A base agent with more satisfying performance in the local region of a state has more influence on the final decision. As a result, a base agent is not evaluated solely by its general performance.

The flow chart of our proposed framework is illustrated in Fig. 1. l diverse base agents ρ_i are trained in the same environment ξ, where $i = 1, 2, ..., l$. The diversity of the agents can be caused by using different training algorithms, structures, and parameters of a model. The DQN is selected as a base agent for our framework demonstration due to its popularity and efficacy. $\rho_i(s)$ maps a state s to an action.

An evaluation set $\mathcal{V}$ is collected for dynamic evaluation, which is discussed in Sect. 3.1 and the performance of ρ_i on $s \in \mathcal{V}$ ($U(s, \rho_i)$) is evaluated according to the n-step cumulative reward. As the evaluation process is independent from the new state, it can be carried out offline. A more detailed explanation is discussed in Sect. 3.2. Given a test state s_t at t, its similarity to $s \in \mathcal{V}$ is measured according to ρ_i by $sim(s, s_t, \rho_i)$ (Sect. 3.3). The k most similar $s \in \mathcal{V}$ to s_t are then chosen to form a similar set $\mathcal{I}_{(s_t,\rho_i)}$. The local competence of ρ_i on s_t ($LC(\rho_i, s_t)$) is then calculated according to $U(s, \rho_i)$, where $s \in \mathcal{I}_{(s_t,\rho_i)}$, as discussed in Sect. 3.4. The weight of ρ_i (W_i) is updated at every v time-interval according to $LC(\rho_i, s_t)$, where t is the time for update (Sect. 3.5). Finally, the time complexity of our model is discussed in Sect. 3.6.

To apply our proposed ensemble method in an application, l base agents should be first trained until convergence, where l can be determined according to the constraints and the difficulty of the application. An evaluation set $\mathcal{V}$ is then generated by randomly selecting states from the interaction between each base agent and the environment. The performance of the base agents are evaluated according to their reward achieved in the next n steps starting initially at a state in $\mathcal{V}$. During the test phase, the weights of the base agents are updated periodically according to their local competence on the most similar states in $\mathcal{V}$ to a test state. The decisions of a base agent on a test state is combined by the weighted average method.

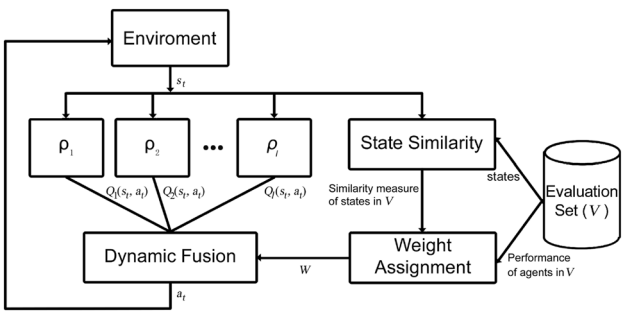

Fig. 1 The flow chart of our proposed dynamic fusion method

3.1 Evaluation set

Similar to the dynamic fusion methods [6, 15] used in supervised and unsupervised learning, an evaluation set $\mathcal{V}$ is collected to evaluate the local competence of an agent in our method. However, because $\mathcal{V}$ is not available in RL, it should be generated by interactions between the agent and the environment. Samples in $\mathcal{V}$ should be able to evenly cover the distribution of states in an application. Considering the differing abilities of base agents, each agent contributes to the generation of $\mathcal{V}$ in our method. For each ρ_i, $|\mathcal{V}|/l$ states are randomly selected as initial states in e episodes, where $|\mathcal{V}|$ is the cardinality of $\mathcal{V}$ and l is the number of base agents. All initial states chosen by all base agents form $\mathcal{V}$.

In addition, the size of $\mathcal{V}$ is also an important factor. According to the law of large numbers, the distribution of a larger $\mathcal{V}$ is closer to the distribution of the actual state space. However, a larger $\mathcal{V}$ also increases the time complexity of evaluation since each state in $\mathcal{V}$ is measured with a test state for each base agent in evaluation. The influence of $\mathcal{V}$ is discussed experimentally in Sect. 4.

3.2 Performance evaluation

The performance of an agent can only be estimated as no supervision is available in RL. Given a state s_t, the performance of ρ_i on a given state s_t is defined as the expected accumulative reward in the next n steps with a discount factor, defined as:

$$U(s_t, \rho_i) = \mathbb{E}(\Sigma_{i=1}^{n}(\gamma^i R_{t+i}^{\rho_i})) \tag{2}$$

where γ denotes the discount factor, and $R_{t+i}^{\rho_i}$ denotes the immediate reward obtained by ρ_i in s_{t+i}. The expected value is considered due to the non-deterministic transaction function in RL. n determines the number of future rewards related to s_t. Generally, a small n is not preferable since observation on s_t may not be enough to achieve reasonable estimation, *i.e.,* the values of U for most states may be similar. However, when n is large, the accumulative reward may not be closely related to the current decision.

3.3 State similarity

The similarity between a pair of states is measured in the latent feature space of a base agent. Specifically, the distance measurement is carried out in the space of the last convolutional layer of the CNN in the DQN. The reasoning behind using the latent space rather than the input space is that the latent space is more related to the actual decision. The decisions of an agent on states close to each other in the latent space are expected to be similar. Let $\phi_i(s)$ be the output of the last convolutional layer of ρ_i on s. The similarity of the two states (s_1 and s_2) is defined as:

$$sim(s_1, s_2, \rho_i) = dist(\phi_i(s_1), \phi_i(s_2)) \tag{3}$$

where *dist* is a distance function and the 2-norm distance measure is used in this study.

3.4 Local competence estimation

The local competence, denoted by $LC(\rho_i, s_t)$, describes the ability of ρ_i in the region around s_t, and is estimated by U in (2) and *sim* in (3) using the validation set $\mathcal{V}$.

Given an unseen state s_t, the k-nearest neighbors of s_t are selected from $\mathcal{V}$ to form the similar state set $\mathcal{I}_{(\rho_i)}$ according to (3) of ρ_i. To have a fair evaluation on each agent, the final similar state set $\mathcal{I}$ is calculated as the union of $\mathcal{I}_{(\rho_i)}$ for all base agents used in the evaluation, where $|\mathcal{I}| \geq k$. k controls the size of $\mathcal{I}$. A small $\mathcal{I}$ may not be sufficient to obtain an accurate estimation on local competence. On the other hand, a large value of k may increase the difference between s_t and states in $\mathcal{V}$, so that the measure does not focus on a local region.

$LC(\rho_i, s_t)$ is then defined as the average U of ρ_i on each state in $\mathcal{I}$ as follows:

$$LC(\rho_i, s_t) = \mathbb{E}_{s \in \mathcal{I}}(U(s, \rho_i)) \tag{4}$$

As states in $\mathcal{I}$ are closely related to s_t, better performance is achieved by ρ_i on $\mathcal{I}$, and a higher local competence for s_t is expected.

3.5 Dynamic weight assignment

The weight for s_t ($w(\rho_i, s_t)$) is calculated to ρ_i according to $LC(\rho_i, s_t)$, defined as:

$$w(\rho_i, s_t) = \frac{LC(\rho_i, s_t)}{\sum_{i=1}^{l} LC(\rho_i, s_t)} \tag{5}$$

As consecutive states are highly dependent in RL, a sudden change of weights in base agents should be avoided. The new weight of ρ_i (W_i') is adjusted by $w(\rho_i, s_t)$ from the previous weight W_i as follows:

$$W_i' = \begin{cases} w(\rho_i, s_t) & t = 0 \\ \alpha w(\rho_i, s_t) + (1 - \alpha) W_i & t > 0 \end{cases} \tag{6}$$

where α denotes the update parameter, which indicates the degree of the influence of $w(\rho_i, s_t)$ on W_i.

The weight of a base agent is updated at every v time-interval, where $v \in \mathcal{Z}^+$. A small v causes frequent updates on the weight, which increases the time complexity. Moreover, it may also cause unstable performance of our model since

decisions in consecutive states should be closely related and generally the bias on base classifiers should be changed in a short period of time. On the other hand, the weight values may not accurately reflect the ability of base agents on the current states when v is too large. Therefore, as v plays an important role in our model, its value should be chosen carefully.

The final $\widetilde{Q}(s_t, a_t)$ for decision-making is obtained based on the following formula:

$$\widetilde{Q}(s_t, a_t) = \sum_{i=1}^{l} \mathcal{W}_{t,i} \, \text{softmax}(Q_i(s_t, a_t)) \tag{7}$$

In order to prevent Q_i's range from affecting the final decision, we used softmax on Q_i before fusion. The algorithm will select the action a_t with the highest $\widetilde{Q}(s_t, a_t)$ value.

3.6 Time complexity

In general, dynamic fusion uses more time than static fusion. Every time the dynamic weights are calculated, our method needs to calculate the distance between the current state s_t and every other state in the evaluation set $\mathcal{V}$ to find the nearest neighbor subset $\mathcal{I}_{(\rho_i)}$ in the l feature spaces. Therefore the time complexity is $O(kl|\mathcal{V}|)$. Although this method has a higher time complexity than static fusion, the metric of distance can be computed in parallel, which means that the calculation time can be reduced by parallel computation in some applications with high real-time requirements.

4 Experiment

This section aims to experimentally evaluate and compare our proposed dynamic fusion algorithm with the static fusion methods in three Atari games, *i.e.,* Seaquest, SpaceInvaders, and Breakout. The experimental settings are discussed in Sect. 4.1. The comparative analysis of our dynamic fusion algorithm with other static methods and base agents is given in Sect. 4.2. The analysis on the parameters and state similarity are given in Sects. 4.3 and 4.4. Finally, the running time is discussed in Section 4.5

4.1 Experimental setting

Three Atari games provided by the Arcade Learning Environment (ALE) [2] are chosen. We follow the standard experimental settings [23], in which each observation is a screen capture resized to 84×84 pixels in gray-scale. A state is defined as the latest four observations.

The standard static fusion methods, including majority voting (*vote*) [14], average (*avg*) [1], and weighted average (*wavg*) [9], are compared with our proposed method. The weights of base agents in *wavg* rely on the average accumulative reward in ten episodes in the evaluation:

$$W_{wavg}(\rho_i) = \frac{R(\rho_i)}{\sum_{i=1}^{l}(R(\rho_i))} \tag{8}$$

where $R(\rho_i)$ denotes the accumulative reward of ρ_i running ten episodes. For our method, the size of the evaluation set ($\mathcal{V}$) is set to 5,000. In order to have a stable evaluation on an agent on a state, n is set to 50 in the applications, where n is the number of steps in which the rewards obtained by an agent are used in evaluation. The award discount factor γ is 0.99. k indicates the amount of the most similar states in the evaluation set to a new state, is set as $\{20, 30, 40, 50, 60\}$. The tradeoff parameter and weight update interval are set as $\alpha \in \{0.1, 0.5, 1\}$ and $v \in \{10, 50, 100\}$ respectively.

A DQN [23] is selected as the base agent. Six base agents with different network structures, as shown in Table 1 are trained in order to increase the diversity. The softmax function is used to reduce the influence of the value domain of the Q values of different models in all the agents. An ensemble system with more base agents generally achieves better performance [14], but will also increase the time and space complexity. The size of the agent pool should be determined according to resource limitation and difficulty of the application. In our experimental setting, four out of these six agents are selected randomly to form an ensemble model in different applications. The setting details of the base agents are shown in Table 2. The other training parameters of the

Table 1 The structures of the base agents

Models	Convolutional layers	Full connection layers
$model_1$	8×8 conv 32/4	FC-512
	4×4 conv 64/2	FC-action number
	3×3 conv 64	
$model_2$	8×8 conv 32/4	FC-512
	4×4 conv 64	FC-action number
$model_3$	8×8 conv 32/4	FC-512
	4×4 conv 64/2	FC-action number
	3×3 conv 64	
	2×2 conv 64	
$model_4$	7×7 conv 16/4	FC-256
	3×3 conv 64	FC-action number
$model_5$	8×8 conv 32/4	FC-512
	4×4 conv 64/2	FC-1024
	3×3 conv 64	FC-256
	2×2 conv 64	FC-action number
$model_6$	8×8 conv 32 /4	FC-1024
	4×4 conv 64/2	FC-256
	3×3 conv 64	FC-action number
	2×2 conv 64	

DQNs are set as the ones in [23]. The average accumulated rewards of 50 independent runs are used as evaluation.

4.2 Performance comparison

The dynamic fusion method is compared with the base agents and other fusion methods, and the results are presented in a box plot shown in Fig. 2. The y-axis indicates the accumulated reward. The red lines and green triangles represent the median and mean respectively. The top and bottom lines denote the maximum and minimum values, while the top and bottom lines of the rectangle denote the upper and lower quartiles. The circle denotes the outlier value.

An ensemble method is less stable than a single agent since an ensemble method has more freedom, *i.e.,* the quartile size of ensemble methods is larger. The degree of fluctuation of the proposed dynamic fusion is more than in static fusion on average since the weights of base agents are evaluated at each time interval. *Our method* is more stable than *vote*, *avg*, and *wavg* in Spaceinvaders, while in Seaquest and Breakout, *our method* has a higher degree of fluctuation than static fusion methods.

The average accumulated rewards of ensemble methods are always higher than their base agents with the exception of *vote* and *avg* in SpaceInvaders, *i.e.,* the ensemble method generally outperforms its base agents. *vote* has the worst performance among *avg*, *wavg*, and *our method*. One reason for this poor performance is that *vote* simply chooses the most selected action, but ignores the degree of confidence on each action estimated by a base agent. Higher average accumulated rewards achieved by *wavg* in comparison with *avg* indicates that the static evaluation on a base agent is reasonable. *Our method* achieves the highest average rewards among all methods in all applications, which confirms that the local competence of a base agent is evaluated reasonably in our method. Dynamical weight assignment to base agents can improve the performance of the ensemble system.

Table 2 Models and training episodes of base agents in the applications

		Seaquest	Spaceinvaders	Breakout
ρ_1	model	$model_1$	$model_1$	$model_3$
	episodes	2,500,000	15,000,000	2,000,000
ρ_2	model	$model_2$	$model_2$	$model_4$
	episodes	2,500,000	15,000,000	2,000,000
ρ_3	model	$model_5$	$model_3$	$model_5$
	episodes	2,500,000	25,000,000	4,000,000
ρ_4	model	$model_6$	$model_4$	$model_6$
	episodes	2,500,000	25,000,000	2,000,000

4.3 Parameter discussion

Our proposed dynamic fusion method with different parameter settings, including α, k, and v, is compared with the best static fusion algorithm in each application, *i.e., wavg* in Seaquest and SpaceInvaders, and *avg* in Breakout, where α denotes the update rate of weights of base agents, k indicates the number of most similar validation states to a new state, and v represents the time interval of the weight update. The values in Fig. 3 represent the difference between the average reward achieved by our method with a particular parameter setting and the best static fusion algorithm. A darker red or blue cell indicates how much better or worse our method performs compared with the best static method.

In Seaquest, the performance of *our method* decreases with the increase of v, while the inverse can be observed in SpaceInvaders in which *our method* with a large v obtains better rewards on average. For Breakout, modifying the size of v is insignificant and the highest average reward is achieved when $v = 10$. A faster weight update for base agents is required in Seaquest since the strategy of the submarine needs to be altered when fishes materialize differently on the screen. A small v causes the RL system to be more effective at detecting the sudden appearance of fishes and adjusting its policy accordingly. However, the invaders do not appear suddenly and their positions are relatively stable in SpaceInvaders. Therefore, no frequent adjustment on policy is needed. The RL

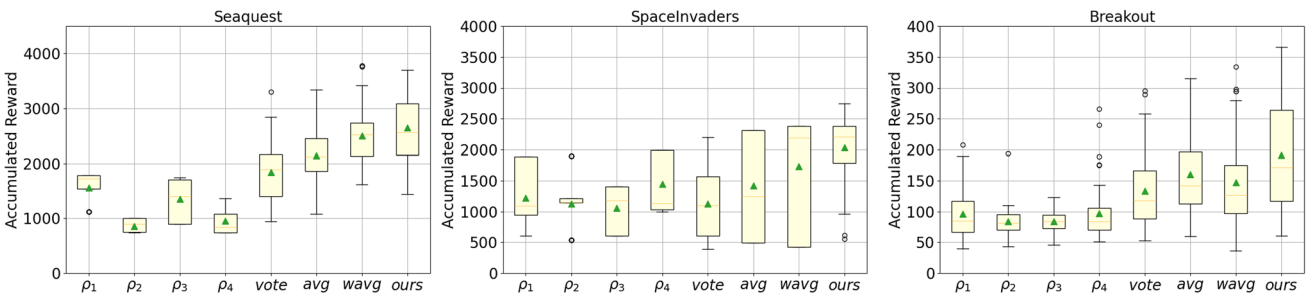

Fig. 2 Performance comparison of base agents, *vote*, *avg*, *wavg*, and our method in terms of accumulated reward

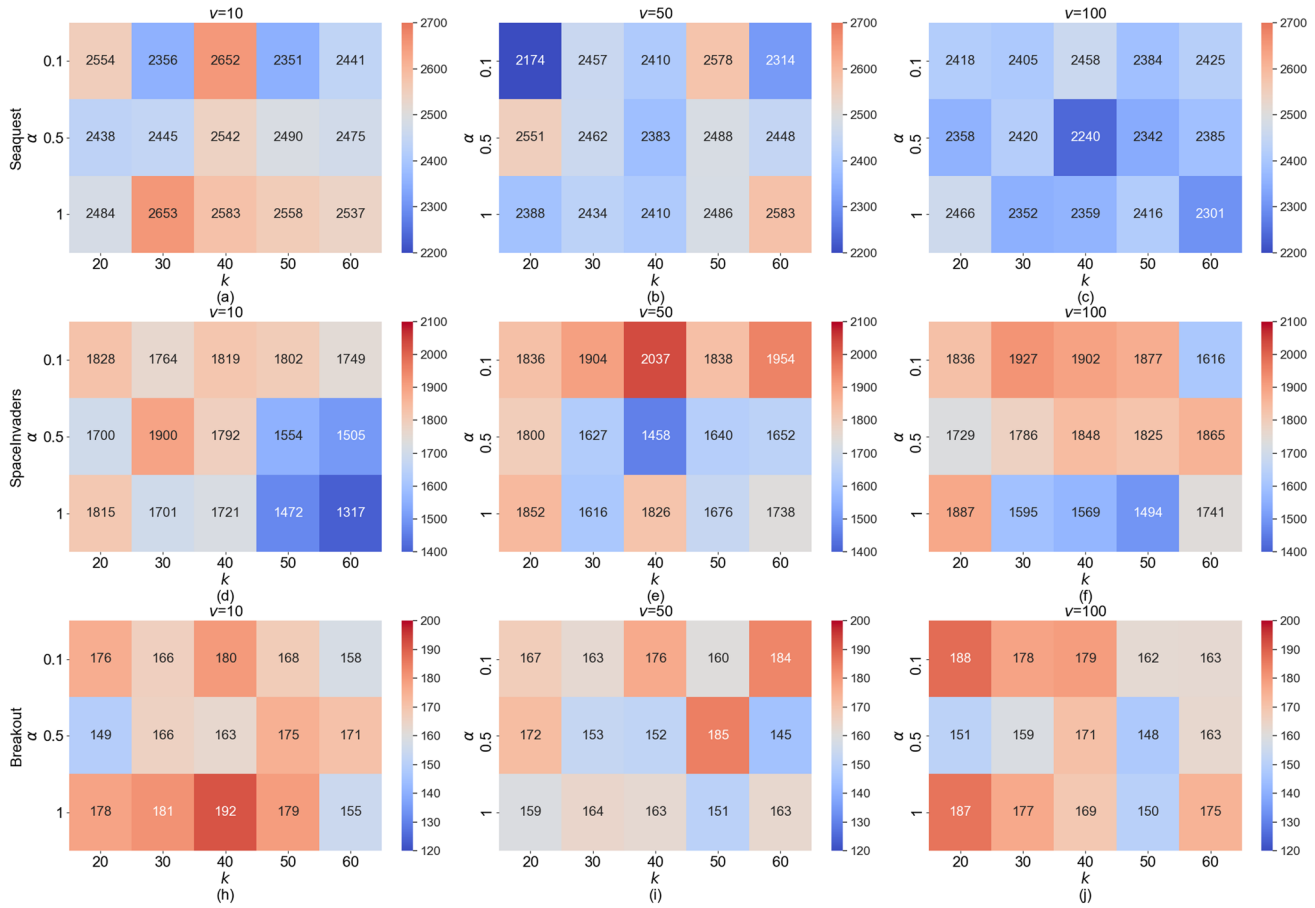

Fig. 3 Performance comparison of dynamic fusion algorithm and best other fusion method in terms of average accumulated reward

system with either fast ($v = 10$) or slow ($v = 100$) policy updates performs well in Breakout. This is most likely due to the unchanging bricks and generally stable ball trajectory causing the agent to perform well with slower weight updates, but also take advantage of fast weight updates when the ball hits the brick corner and bounces back unpredictably.

In conclusion, Seaquest needs frequent updates but SpaceInvaders does best with slow updates. This explains why *our method* with larger and smaller values of *alpha* are preferable in Seaquest and SpaceInvaders respectively. For Breakout, *our method* with either $v = 10$ or 100 has good performance when α is equal to either 0.1 or 1, respectively.

k should be not be too large or small since a small k leads to a biased evaluation, while a large k may cause validation samples that are not similar to the new state. The experimental results confirm that the performance of our dynamic fusion method with the best combination of α and v achieves the best performance when $k = 30$ and 40 in these three applications.

4.4 State similarity analysis

The similarity in the last output of the feature map of the CNN is investigated in this section. A state is chosen as an example from each application and is shown in Fig. 4. Although states are converted to gray-scale before use, they're shown in color for better visibility. Each column shows a state which contains four observations, *i.e.*, $s = \{O_1, O_2, O_3, O_4\}$ and each observation is an 84×84 image. Given a source state (*source*), the nearest states from a validation set are chosen in the latent space of the CNN (*ours*) and the input feature space (*input*).

In Seaquest, the position of the submarine in *ours* is closer to the one in *source* compared with *input*. Moreover, *input* does not contains fishes in the upper left corner. In SpaceInvaders, no obvious difference between *ours* and *input* can be observed. It may be because the decisions on SpaceInvaders are closely related to its original input screen. The bricks dominate the similarity in *input* in Breakout, *i.e.*, the shapes of bricks in *input* is the same as *source*. However, *ours* focuses on the similarity of the plate locations and ball

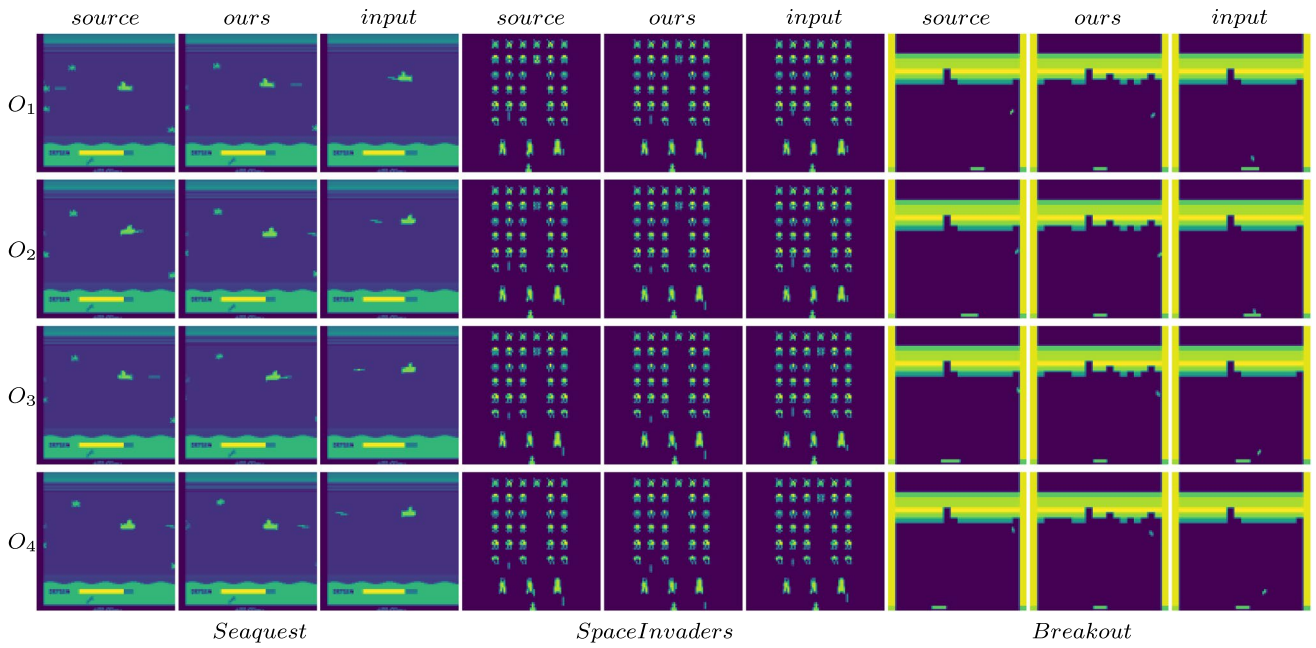

Fig. 4 Illustration of the most similar state measured in the input (*input*) and latent (*ours*) space

Table 3 Average times (in second) of making a decision on a test state of *vote*, *avg*, *wavg*, and *our method*

	Seaquest	SpaceInvaders	Breakout
vote	0.021s	0.042s	0.019s
avg	0.021s	0.042s	0.019s
wavg	0.021s	0.042s	0.019s
our method	1.402s	2.524s	1.401s

trajectory. These examples illustrate that the similarity measured in the latent space of the CNN is more closely related to the decision of an agent compared with the one in the input space.

4.5 Running time comparison

The average decision making time on a test state of the methods is shown in Table 3. It is clear that the dynamic fusion algorithm spends more time than static methods, which is consistent with the studies in classification [6]. This is because dynamic weight assignment during the test phase includes the similarity measure between a new state and the states in the validation set. Time is sacrificed to gain a better accumulative reward in our method. As the weight has not been updated for static methods, their decision times are the same. SpaceInvaders requires longer decision time than the other two applications since the models used in SpaceInvaders learned a higher dimensional representation in the last convolutional layer, thus increasing time spent in the state comparison stage.

5 Conclusion

Utilizing the full ability of agents is a key problem in ensemble. The current studies on DRL ensemble focus on static fusion which assumes a base agent performs consistently for all scenarios, which may not be realistic in some applications. These methods only capture the general performance and ignore the capacity of an agent in a particular state. This assumption means that the ensemble systems cannot take full advantage of all base agents. This study proposes the first dynamic fusion method for DRL in order to fully utilize the ability of each base agent for different states. The weights are dynamically assigned to agents according to their local competence on the region around a test state during the test phase. A base agent is evaluated according to the states in the evaluation set collected by interaction between an agent and environment in terms of the rewards obtained in the next n steps. The similarity of a new state and a validation state is measured by Euclidian distance in the latent space of an agent. The weight of a base agent is updated in intervals according to its performance on validation states and the similarity between validation states and a new state. As a result, the importance of a base agent when making decisions will depend on its local competence on a test state , as opposed to the overall performance of the agent on all states. Experimental results confirm that the weight

assignment by our dynamic method successfully utilizes the ability of agents in three different applications. Our method sacrifices execution time to gain a higher reward. The results confirm that the performance of our method is better than its base agents and the static methods.

The validation set plays an important role in our method since it has been used to evaluate the local competence of an agent. Moreover, its size affects the execution time since the similarity is measured for each validation state and test state. How to choose a more representative validation state is a possible extension of this study. The validation samples can be chosen aiming at minimizing the difference between the distribution of the validation set and the real one.

Acknowledgements This paper is supported by the Natural Science Foundation of Guangdong Province, China (No. 2018A030313203) and the Fundamental Research Funds for the Central Universities (No. 2018ZD32).

References

1. Anschel O, Baram N, Shimkin N (2017) Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In: International conference on machine learning, pp 176–185
2. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: an evaluation platform for general agents. J. Artificial Intell. Res. 47:253–279
3. Bonaccio S, Dalal RS (2006) Advice taking and decision-making: an integrative literature review, and implications for the organizational sciences. Organizational behavior and human decision processes 101(2):127–151
4. Brys T, Harutyunyan A, Vrancx P, Nowé A, Taylor ME (2017) Multi-objectivization and ensembles of shapings in reinforcement learning. Neurocomputing 263:48–59
5. Buckman J, Hafner D, Tucker G, Brevdo E, Lee H (2018) Sample-efficient reinforcement learning with stochastic ensemble value expansion. In: Advances in Neural Information Processing Systems, pp 8224–8234
6. Chan PP, Yeung DS, Ng WW, Lin CM, Liu JN (2012) Dynamic fusion method using localized generalization error model. Inform Sci 217:1–20
7. Chan PP, Wang YX, Yeung DS (2020) Adversarial attack against deep reinforcement learning with static reward impact map. In: ACM ASIACCS, InPress
8. Chen L, Lu L, Feng K, Li W, Song J, Zheng L, Yuan Y, Zeng Z, Feng K, Lu W et al (2009) Multiple classifier integration for the prediction of protein structural classes. J Comput Chem 30(14):2248–2254
9. Xl Chen (2018) Cao L, Li Cx, Xu Zx, Lai J (2018) Ensemble network architecture for deep reinforcement learning. Mathematical Problems in Engineering
10. Cruz RM, Sabourin R, Cavalcanti GD (2018) Dynamic classifier selection: recent advances and perspectives. Inform Fusion 41:195–216
11. Dietterich TG (2000) Ensemble methods in machine learning. In: International workshop on multiple classifier systems, Springer, pp 1–15
12. Duell S, Udluft S (2013) Ensembles for continuous actions in reinforcement learning. In: ESANN
13. Faußer S, Schwenker F (2011) Ensemble methods for reinforcement learning with function approximation. In: International workshop on multiple classifier systems, Springer, pp 56–65
14. Faußer S, Schwenker F (2015) Neural network ensembles in reinforcement learning. Neural Process Lett 41(1):55–69
15. Feng X, Xiao Z, Zhong B, Qiu J, Dong Y (2018) Dynamic ensemble classification for credit scoring using soft probability. Appl Soft Comput 65:139–151
16. Gao Z, Gao Y, Hu Y, Jiang Z, Su J (2020) Application of deep q-network in portfolio management. In: 2020 5th IEEE international conference on big data analytics (ICBDA), IEEE, pp 268–275
17. Gosavi A (2004) A reinforcement learning algorithm based on policy iteration for average reward: empirical results with yield management and convergence analysis. Mach Learning 55(1):5–29
18. Gu S, Holly E, Lillicrap T, Levine S (2017) Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, pp 3389–3396
19. Hans A, Udluft S (2010) Ensembles of neural networks for robust reinforcement learning. In: 2010 ninth international conference on machine learning and applications, IEEE, pp 401–406
20. Hessel M, Modayil J, Van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2018) Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-second AAAI conference on artificial intelligence
21. Huang W, Chan PP, Zhou D, Fang Y, Liu H, Yeung DS (2016) Multiple classifier system with sensitivity based dynamic weighting fusion for hand gesture recognition. In: 2016 international conference on wavelet analysis and pattern recognition (ICWAPR), IEEE, pp 31–36
22. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602
23. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
24. Saleena N et al (2018) An ensemble classification system for twitter sentiment analysis. Proc Comput Sci 132:937–946
25. Sallab AE, Abdou M, Perot E (2017) Yogamani S (2017) Deep reinforcement learning framework for autonomous driving. Electron Imaging 19:70–76
26. Saphal R, Ravindran B, Mudigere D, Avancha S, Kaul B (2020) Seerl: Sample efficient ensemble reinforcement learning. arXiv preprint arXiv:200105209
27. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: International conference on machine learning, pp 1889–1897
28. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484–489
29. Soares SG, Araújo R (2016) An adaptive ensemble of on-line extreme learning machines with variable forgetting factor for dynamic system prediction. Neurocomputing 171:693–707
30. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Thirtieth AAAI conference on artificial intelligence
31. Wang XZ, Xing HJ, Li Y, Hua Q, Dong CR, Pedrycz W (2014) A study on relationship between generalization abilities and fuzziness of base classifiers in ensemble learning. IEEE Trans Fuzzy Syst 23(5):1638–1654
32. Watkins CJCH (1989) Learning from delayed rewards. PhD thesis, King's College, Cambridge

33. Wiering MA, Van Hasselt H (2008) Ensemble algorithms in reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38(4):930–936
34. Wu J (2020) Li H (2020) Deep ensemble reinforcement learning with multiple deep deterministic policy gradient algorithm. Mathematical Problems in Engineering
35. Yeung DS, Chan PP (2009) A novel dynamic fusion method using localized generalization error model. 2009 IEEE international conference on systems. Man and Cybernetics, IEEE, pp 623–628
36. Yu H, Chen Y, Lingras P, Wang G (2019) A three-way cluster ensemble approach for large-scale data. Intl J Approximate Reasoning 115:32–49
37. Yu Z, Li L, Liu J, Zhang J, Han G (2015) Adaptive noise immune cluster ensemble using affinity propagation. IEEE Trans Knowl Data Eng 27(12):3176–3189
38. Zhu Y, Mottaghi R, Kolve E, Lim JJ, Gupta A, Fei-Fei L, Farhadi A (2017) Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, pp 3357–3364