Applied Intelligence (2020) 50:2400–2410
https://doi.org/10.1007/s10489-020-01679-3

Reinforcement learning with convolutional reservoir computing

Hanten Chang[1] · Katsuya Futagami[1]

Published online: 7 March 2020

Abstract
Recently, reinforcement learning models have achieved great success, mastering complex tasks such as Go and other games with higher scores than human players. Many of these models store considerable data on the tasks and achieve high performance by extracting visual and time-series features using convolutional neural networks (CNNs) and recurrent neural networks respectively. However, these networks have very high computational costs because they need to be trained by repeatedly using the stored data. In this study, we propose a novel practical approach called reinforcement learning with a convolutional reservoir computing (RCRC) model. The RCRC model uses a fixed random-weight CNN and a reservoir computing model to extract visual and time-series features. Using these extracted features, it decides actions with an evolution strategy method. Thereby, the RCRC model has several desirable features: (1) there is no need to train the feature extractor, (2) there is no need to store training data, (3) it can take a wide range of actions, and (4) there is only a single task-dependent weight matrix to be trained. Furthermore, we show the RCRC model can solve multiple reinforcement learning tasks with a completely identical feature extractor.

Keywords Reinforcement learning · Reservoir computing · Evolution strategy · Untrained convolutional neural network

1 Introduction

Recently, reinforcement learning (RL) models have achieved great success, mastering complex tasks such as Go [1] and other games [2–4] with higher scores than human players. Many of these models use convolutional neural networks (CNNs) to extract visual features directly from the environment state images [5]. Some models use recurrent neural networks (RNNs) to extract time-series features and achieved higher scores [6].

However, these deep neural networks (DNNs) based models are often very computationally expensive in that they train networks weights by repeatedly using a large volume of past playing data and task-rewards. Certain techniques can alleviate these costs, such as the distributed approach [3, 7] which efficiently uses multiple agents, and the prioritized experienced replay [8] which selects samples that facilitate training. However, the cost of a series of computations, from data collection to action determination, remains high.

The world model [9] can also reduce computational costs by completely separating the training processes between the feature extraction model and the action decision model. The world model trains the feature extraction model in a rewards-independent manner by using variational auto-encoder (VAE) [10, 11] and mixture density network combined with an RNN (MDN-RNN) [12]. After extracting the environment state features, it uses an evolution strategy method called the covariance matrix adaptation evolution strategy (CMA-ES) [13, 14] to train the action decision model. The world model can achieve outstanding scores in famous RL tasks. The separation of these two models results in the stabilization of feature extraction and reduction of parameters to be trained based on task-rewards.

From the success of the world model, it is implied that in the RL feature extraction process, it is important to extract the features that express the environment state sufficiently rather than features trained to get higher rewards. Adopting this idea, we propose a new method called "reinforcement learning with convolutional reservoir computing (RCRC)". The RCRC model is inspired by the reservoir computing.

Reservoir computing [15] is a kind of RNNs, and the model weights are set to random. One of the reservoir computing models, the echo state network (ESN) [16, 17] is used to solve

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10489-020-01679-3) contains supplementary material, which is available to authorized users.

✉ Hanten Chang
s1820554@s.tsukuba.ac.jp

Katsuya Futagami
s1820559@s.tsukuba.ac.jp

1 Division of Policy and Planning Sciences, Faculty of Engineering, Information and Systems, University of Tsukuba, 1-1-1 Tennoudai, Tsukuba, Ibaraki, Japan

time-series tasks such as future value prediction. For this, the ESN extracts features for the input signal based on the dot product of the input signal and fixed random-weight matrices generated without training. Surprisingly, features obtained in this manner are expressive enough to understand the input, and complex tasks such as short-term chaotic time-series prediction can be solved by using them as the input for a linear model. In addition, the ESN has solved various tasks in multiple fields such as time-series classification [18, 19] and Q-learning-based RL [20–22]. Similarly, in image classification, the model that uses features extracted by the CNN with fixed random-weights as the ESN input achieves high accuracy classification with a smaller number of parameters [23].

Based on the success of the fixed random-weight models, the RCRC model extracts the visual features of the environment state using fixed random-weight CNN, and, using these features as the ESN input, extracts time-series features of the environment state transitions. After extracting the environment state features, we use CMA-ES [13, 14] to train a linear transformation from the extracted features to the actions, as in the world model. This model architecture results in the omission of the training process of feature extractor and reduced computational costs; there is also no need to store past playing data. Furthermore, we show that the RCRC model can solved multiple RL tasks with the completely identical structure and weights feature extractor.

Our contributions in this study are as follows:

- We developed a novel and widely applicable approach to extract visual and time-series features of an RL environment state using fixed random-weights networks feature extractor with no training.
- We developed the RCRC model that doesn't need to store data and to train feature extractor.
- We showed that the RCRC model can solve different tasks with training only a single task-dependent weight matrix and using the completely identical feature extractor.

2 Related work

2.1 Reservoir computing

Reservoir computing is one of the RNNs and it extracts features of the input without training for the feature extraction process. In this study, we focus on a reservoir computing model, ESN [16, 17]. The ESN was initially proposed to solve time-series tasks [16] and is regarded as an RNN model [15, 24].

Let the N-length, D_u-dimensional input signal be $u = \{u(1), u(2), ..., u(t), ..., u(N)\} \in \mathbb{R}^{N \times D_u}$ and the signal that added one bias term to input signal be $U = [u; 1] = \{U(1), U(2), ..., U(T), ..., U(N)\} \in \mathbb{R}^{N \times (D_u+1)}$. Let [;] be a vector concatenation. The ESN gets features called the reservoir state $X = \{X(1), ..., X(t), ..., X(N)\} \in \mathbb{R}^{N \times D_x}$ as follows:

$$\widetilde{X}(t+1) = f\left(W^{\text{in}} U(t) + W X(t)\right)$$
$$X(t+1) = (1-\alpha) X(t) + \alpha \widetilde{X}(t+1)$$

where the matrices $W^{\text{in}} \in \mathbb{R}^{(D_u+1) \times D_x}$ and $W \in \mathbb{R}^{D_x \times D_x}$ are sampled from a probability distribution such as a Gaussian distribution, and f is the activation function which is applied element-wise. As the activation function, *linear* and *tanh* functions are generally used; it is also known that changing the activation function according to the task improves accuracy [25, 26]. The leakage rate $\alpha \in [0,1]$ is a hyperparameter that tunes the weight between the current and the previous values, and W has two major hyperparameters called sparsity and spectral radius. The sparsity is the ratio of 0 elements in matrix W and the spectral radius is a memory capacity parameter which is calculated by the maximal absolute eigenvalue of W.

Finally, the ESN estimates the target signal $y = \{y(1), y(2), ..., y(t), ..., y(N)\} \in \mathbb{R}^{N \times D_y}$ as

$$y(t) = W^{\text{out}} [X(t); U(t); 1].$$

The weight matrix $W^{\text{out}} \in \mathbb{R}^{D_y \times (D_x + D_u + 1)}$ is estimated by a linear model such as ridge regression. An overview of reservoir computing is shown in Fig. 1.

The unique feature of the ESN is that the two matrices W^{in} and W are randomly generated from a probability distribution and fixed. Therefore, the training process in the ESN consists only of a linear model to estimate W^{out}, hence the ESN has very low computational cost. In addition, the reservoir state reflects complex dynamics despite being obtained by random matrix transformation, and it is possible to use it to predict complex time-series by a simple linear transformation [16, 27, 28]. Because of the low computational cost and high expressiveness of the extracted features, the ESN is also used to solve other tasks such as time-series classification [18, 19], and image classification [23].

There are also previous works that use the reservoir computing to extract features for Q-learning [20–22]. However, to the best of our knowledge, no research has been extended to use reservoir computing to extract visual features in reinforcement learning tasks, and no research has been combined reservoir computing with evolution strategy. In our proposed method, it is unique that both visual and time-series features can be acquired with no training. Furthermore, by combining evolution strategy, a wide range of actions can be taken naturally, and it gives several desirable features such as no need for storing data.

2.2 World models

The world model [9] is one of the RL models that separates the training of the feature extraction model and the action decision

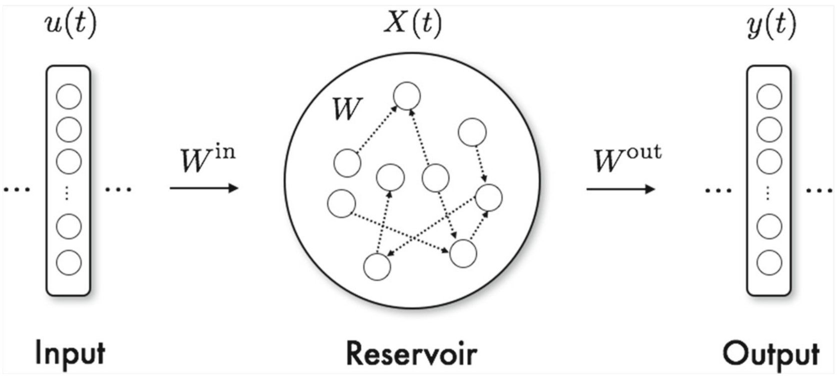


Fig. 1 Reservoir Computing overview for the time-series prediction task

model to train the model more efficiently. It uses VAE [10, 11] and MDN-RNN [12] as feature extractors. They are trained in a reward-independent manner with randomly played 10000 episodes data. As a result, in the feature extraction process, the reward-based parameters are omitted, and there remains a single weight matrix to be trained that decides the action. The weight is trained by CMA-ES [13, 14]. Although the feature extraction model is trained in a reward-dependent manner, the world model achieves outstanding scores in an RL task `CarRacing-v0` [29]. Furthermore, the world model can be trained to predict the next environment state, thus it can generate the environment by itself. By training the action decision model in this self-play environment, the world model solved an RL task `DoomTakeCover-v0` [30, 31].

CMA-ES is one of the evolution strategy methods used to optimize some parameters using a multi-candidate search generated from a multivariate normal distribution. As CMA-ES updates parameters using only the evaluation scores calculated by actual playing, it can be used regardless of whether the actions of the environment are continuous or discrete values [13, 14]. Furthermore, the training can be faster because it can be parallelized by the number of parameter candidates.

The world model reduces the computational cost and accelerates the training process by separating training processes of models and applying CMA-ES. However, in the world model, it is necessary to independently optimize VAE, MDN-RNN, and CMA-ES each other. Furthermore, in optimizing the feature extractor models, they need to save considerable data and be trained by repeatedly using those data.

3 Proposal model

3.1 Basic concept

The results of world model [9] implies that in RL tasks, it requires features that sufficiently express the environment state, rather than features trained to get higher rewards. We thus focus on extracting features that sufficiently express environment states by fixed random-weights networks. Using fixed random-weights networks as feature extractor has some advantages, such as no need for both training feature extractor and storing data, while being able to sufficiently extract features. For example, a simple CNN with fixed random-weights can extract visual features and achieve high accuracy in image classification [23]. Although the MDN-RNN weights are fixed in the world model, it can achieve high performance [32]. In the ESN, the model can predict complex time-series using features extracted by random matrices transformations [16, 27, 28]. Therefore, it can be considered that CNNs can extract visual features and ESN can extract time-series features, even if their weights are random and fixed. From this hypothesis, we propose the RCRC model, which includes both fixed random-weight CNN and ESN.

3.2 Proposal model overview

The RCRC model is composed of three layers: the untrained CNN layer, the reservoir computing layer and the controller layer. In the first layer, it extracts visual features by using a fixed random-weight CNN. In the second layer, it uses transitions of the visual features extracted in the first layer as input to the ESN to extract the time-series features. In the two layers above that collectively called the convolutional reservoir computing layer, visual and time-series features are extracted with no training. In the final layer, a single weight matrix of a linear transformation from the outputs of the convolutional reservoir computing layer to the actions is trained. A model overview is shown in Fig. 2.

In the previous study, there is a similar world model-based approach [33] that uses fixed random-weights in VAE and a LSTM [34]. However, this approach is ineffective in solving `CarRacing-v0` [29]. In the training process, the best average score over 20 randomly created tracks of each generation was less than 200. However, as mentioned further on, we achieved an average score above 900 over 100 randomly created tracks in `CarRacing-v0` by taking the reservoir computing knowledge in the RCRC model. We also solved `DoomTakeCover-v0` [30, 31] by using the convolutional reservoir computing layer whose structure and weights are completely identical in `CarRacing-v0`.

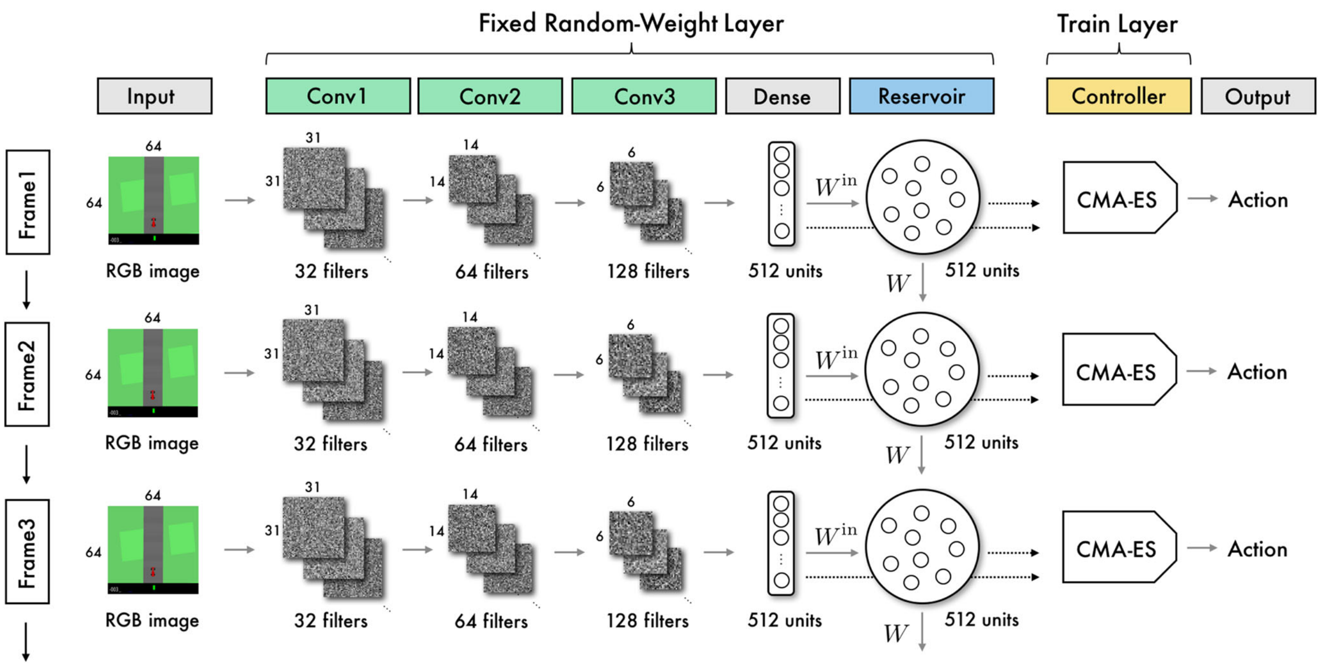


Fig. 2 RCRC overview to decide the action for `CarRacing-v0`: the feature extraction layer are called the convolutional reservoir computing layer, and the weights are sampled from Gaussian distributions and then fixed. For `DoomTakeCover-v0` we use the completely identical convolutional reservoir computing layer in `CarRacing-v0`. In both tasks, we train only a single weight in the controller layer to take task-dependent actions

The characteristics of the RCRC model are as follows:

– The computational cost of the RCRC model is very low because visual and time-series features of environment states are extracted using a convolutional reservoir computing layer whose weights are fixed and random.
– In the RCRC model, only a single weight matrix in the controller layer needs to be trained because the feature extraction model (the convolutional reservoir computing layer) and the action training model (the controller layer) are separated.
– The RCRC model can take a wide range of actions regardless of continuous or discrete, because the model training process is based on the scores measured by actual playing.
– Past data storage is not required, as neither the convolutional reservoir computing layer nor the controller layer needs to be trained by the past data as in back-propagation.
– The convolutional reservoir computing layer can be applied to other tasks without further training for feature extractor because the layer's weights are fixed with task-independent random weights.

3.3 Convolutional reservoir computing layer

In the convolutional reservoir computing layer, the visual and time-series features of the environment state images are extracted by a fixed random-weight CNN and an ESN-based method which has fixed random-weights, respectively. A study using each image's features that are extracted by the fixed random-weight CNN as input to the ESN has been previously conducted, and has shown its ability to classify MNIST dataset [35] with high accuracy [23]. Based on this, we developed a novel and practical approach to solve various RL tasks. By taking advantage of the RL characteristic that the current environment state and the action determine the next environment state, the RCRC model updates the reservoir state with current and previous environment state features. This updating process enables the reservoir state to have time-series features.

More precisely, consider the D_{conv}-dimensional visual features extracted by the fixed random-weight CNN for t-th environment state image $X_{\mathrm{conv}}(t)\in\mathbb{R}^{D_{\mathrm{conv}}}$ and the D_{esn}-dimensional reservoir state $X_{\mathrm{esn}}(t)\in\mathbb{R}^{D_{\mathrm{esn}}}$. The reservoir state X_{esn} is time-series features and updated as follows:

$$\widetilde{X}_{\mathrm{esn}}(t+1) = f\left(W^{\mathrm{in}}X_{\mathrm{conv}}(t) + WX_{\mathrm{esn}}(t)\right)$$
$$X_{\mathrm{esn}}(t+1) = (1-\alpha)X_{\mathrm{esn}}(t) + \alpha\widetilde{X}_{\mathrm{esn}}(t+1).$$

This updating process has no training necessity, and is very fast, because W^{in} and W are random matrices sampled from the probability distribution and fixed.

3.4 Controller layer

The controller layer decides actions by using the output of the convolutional reservoir computing layer, X_{conv} and X_{esn}. Let t-th environment state input vector which added one bias term be $S(t) = [X_{\text{conv}}(t); X_{\text{esn}}(t); 1] \in \mathbb{R}^{D_{\text{conv}}+D_{\text{esn}}+1}$. We suppose that the feature $S(t)$ has sufficient expressive information about the environment states and it can take action by a linear combination of $S(t)$. Therefore, we obtain action $A(t) \in \mathbb{R}^{N_{\text{act}}}$ as follows:

$$\widetilde{A}(t) = W^{\text{out}} S(t)$$
$$A(t) = g(\widetilde{A}(t))$$

where $W^{\text{out}} \in \mathbb{R}^{(D_{\text{conv}}+D_{\text{esn}}+1)\times N_{\text{act}}}$ is the weight matrix and the scalar N_{act} is the number of actions in the task; g is the function which adjusts $\widetilde{A}(t)$ into feasible action space range.

Because the weights of the convolutional reservoir computing layer are fixed, only the weight matrix W^{out} requires training. We optimize W^{out} by using CMA-ES, as in the world model. Therefore, it is possible to parallelize the training process and handle both discrete and continuous values as actions [13, 14]. The process of optimizing W^{out} by CMA-ES are followings:

1. Generate each solution candidate $W_i^{\text{out}}(i = 1, ..., n)$ from a multivariate normal distribution $\mathcal{N}(m, \sigma^2 C)$.
2. Create n environments and workers. Each worker worker$_i$ ($i = 1,...,n$) implements the RCRC model and W_i^{out} is set to the controller layer.
3. In each environment, each worker$_i$ plays m episodes and in each episode, worker$_i$ receives a score $G_{i,j}$ ($j = 1,...,m$).
4. Update evolution paths with the score of each W_i^{out} which is calculated by $G_i = 1/m\sum_{j=1}^{m} G_{i,j}$.
5. Update m, σ, C by using evolution paths.
6. Repeat 1 to 5 until the convergence condition is satisfied or the specified number of repetitions are completed.

In this process, n means the number of solution candidates W^{out} generated at each step. Each worker extracts features, takes the action in each independent environment, and obtains scores.

4 Experiments

4.1 Experiments environments

We evaluate the RCRC model in two famous RL tasks: `CarRacing-v0` [29] in OpenAI Gym [36] and `DoomTakeCover-v0` [30, 31] in ViZDoom [37]. `CarRacing-v0` is a continuous action task and `DoomTakeCover-v0` is a discrete action task. These two tasks are known to improve accuracy by extracting time series features[9]. In both environments, we use the identical structure and weights convolutional reservoir computing layer as a feature extractor to evaluate the generalization ability and train only a single weight in the controller layer.

4.2 CarRacing-v0

`CarRacing-v0` [29] is a car racing game environment that is known as a difficult continuous action task [9]. The goal of this game is to go around the course without getting out by operating a car with three continuous actions: steering wheel, accelerator, and brake. The course is filled with tiles as shown in Fig. 3a. Each time the car passes a tile on the course, $1000/N$ is added to the score. The scalar N is the total number of tiles on the course. The course is randomly generated every time, and the total number of tiles in the course varies around 300. If all the tiles are passed, the total reward will be 1000, but it is subtracted by 0.1 for each frame. The episode ends when all the tiles are passed or when 1000 frames are played. If the player can pass all the tiles without getting out of the course, the reward will be above 900. The definition of "solve" in this game is to get an average of 900 over 100 consecutive trials.

4.3 DoomTakeCover-v0

`DoomTakeCover-v0` [30, 31] is a first-person perspective and 3D vision game. The goal of this game is to survive a long time with avoiding fireballs by operating a player with two discrete actions: move left and move right as shown in Fig. 3b. The fireballs are launched towards the player from the monsters and the survival time steps × 1 will be the score. Player has hit points and can withstand 1 or 2 times of fireballs hit, but if hit points has gone, it will be game over. The screen moves along with its own action, and the fireballs are launched from the backward of the screen toward the player, so it is necessary to recognize the depth of the screen to avoid the fireball. The episode ends when 2100 frames are passed. The definition of "solve" in this game is to get an average of 750 over 100 consecutive trials.

4.4 Procedure

As previously stated, taking advantage of the RCRC model characteristic that feature extractor's weights are task-independent, we use an identical convolutional reservoir computing layer as a feature extractor in both tasks.

We determine hyperparameters based on natural settings without tuning to measure model structure performance. We first resize the environment state image into 64 × 64 with 3 channels pixels and divided by 255 to restrict the each pixel value into [0,1] as input to the convolutional reservoir computing layer. In the convolutional reservoir computing layer, we set 3 convolution layers and 1 dense layer. The filter sizes in the convolutional layers are 31, 14, and 6, and we set the number of the filters to 32, 64, and 128. All strides are set to 2. We set D_{conv} and D_{esn} to 512. The weights in convolution layers are sampled from Gaussian distributions $\mathcal{N}(0, 0.06^2)$; the both weights in the reservoir computing layer W^{in} and W

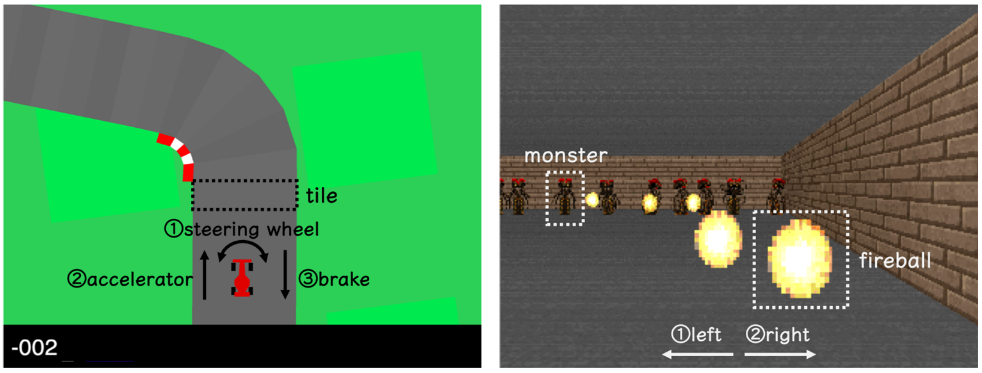


Fig. 3 Example environment state images and actions in each environment

(a) `CarRacing-v0` (b) `DoomTakeCover-v0`

are sampled from Gaussian distributions $\mathcal{N}(0, 0.1^2)$. Regarding the parameters of reservoir computing layer, it is known that sparsity does not significantly affect accuracy, so we set it to 0.8, and we set spectral radius to 0.95 considering that it is often set to be less than 1 [38]. Leakage ratio is set to 0.8 considering it does not have a large effect on accuracy in previous work [23]. All activation functions are set to *tanh* which is often used in the ESN manner. The task-dependent parameters are only W^{out} in the controller layer. Therefore size of W^{out} which is the parameter be trained in a task-dependent manner is 3075 in `CarRacing-v0` and 1025 in `DoomTakeCover-v0`. The examples of the visual features extracted in each convolution layer are shown in Fig. 4.

To get action $A_{\text{car}}(t)$ of `CarRacing-v0`, as in the world model [9], we adjust $\widetilde{A}_{\text{car}}(t)$ which is calculated by dot product of the extracted features and the weight in the controller layer, by the function g_{car} as follows:

$$A_{\text{car}}(t) = g_{\text{car}}(\widetilde{A}_{\text{car}}(t)) = \begin{cases} tanh(\widetilde{A}^{(1)}_{\text{car}}(t)) \\ \left[tanh\left(\widetilde{A}^{(2)}_{\text{car}}(t)\right) + 1.0\right]/2.0 \\ clip\left[tanh\left(\widetilde{A}^{(3)}_{\text{car}}(t)\right), 0, 1\right] \end{cases}$$

where $\widetilde{A}^{(i)}_{\text{car}}$ is i-th value in $\widetilde{A}$ and $clip[x, \lambda_{\min}, \lambda_{\max}]$ is the function that limits the value of x in range from $\lambda_{\min}$ to $\lambda_{\max}$ by clipping. Let $A^{(i)}_{\text{car}}$ be i-th value in A, the values $A^{(1)}_{\text{car}} \in [-1, 1]$, $A^{(2)}_{\text{car}} \in [0, 1]$ and $A^{(3)}_{\text{car}} \in [0, 1]$ are correspond to steering wheel, brake and accelerator, respectively.

To get action $A_{\text{doom}}(t)$ of `DoomTakeCover-v0`, we adjust $\widetilde{A}_{\text{doom}}(t)$ which is calculated by dot product of the extracted features and weight in the controller layer, by the function g_{doom} as follows:

$$A_{\text{doom}}(t) = g_{\text{doom}}(\widetilde{A}_{\text{doom}}(t)) = \begin{cases} \text{left} & (\widetilde{A}^{(1)}_{\text{doom}}(t) \le 0) \\ \text{right} & \left(\widetilde{A}^{(1)}_{\text{doom}}(t) > 0\right) \end{cases}.$$

In the experiments, we use CMA-ES to optimize W^{out} until 500-th generations, and set 16 workers ($n = 16$) which implements the RCRC model for `CarRacing-v0` and 32 ($n = 32$) for `DoomTakeCover-v0`. Each worker is set to simulate over 8 randomly generated trials ($m = 8$), and updates W^{out} with an average of these scores. In optimizing W^{out} in `DoomTakeCover-v0`, we didn't set the max simulation step to evaluate the actual playing ability. As in the world model [9], we evaluate the generalization ability of the models by the average score over 100 randomly created trials. In generalization ability evaluation, we set the weight of the best worker which reached the best average score over 8 trials to the controller layer's weight.

To investigate the ability of network structures, we evaluate three models: the full RCRC model, the RCRC model that removes the reservoir computing layer (visual model), and the RCRC model that has only one dense layer as feature extractor (dense model). The dense model uses flatten vector of 64 × 64 with 3 channels pixels as input and extracts visual features with no convolution. We set the weights of all models to random and fixed. The inputs to the controller layer of the visual model and the dense model are the D_{conv}-dimensional outputs from the dense layer shown in Fig. 2.

5 Results

5.1 Random seed dependency

Since the weight of features extractors in our proposed method are highly depend on random distribution. We first examine how random seed choice affect accuracy in both environments. We use canonical correlation analysis (CCA) [39] to detect linear relationship between two features extracted by different weights decided by different random seeds. In CCA, we calculate maximum canonical correlation, difined by

$$\rho(i,j) = \max_{w_i, w_j} \frac{\text{Cov}\left(w_i^T S^i, w_j^T S^j\right)}{\sqrt{\text{Var}\left(w_i^T S^i\right)\text{Var}\left(w_j^T S^j\right)}}.$$

where Cov(x) is covariance operator and Var(x) is variance operator of x. Let the convolutional reservoir computing layer whose weights are generated by random seed i be CRC$_i$. Thus,

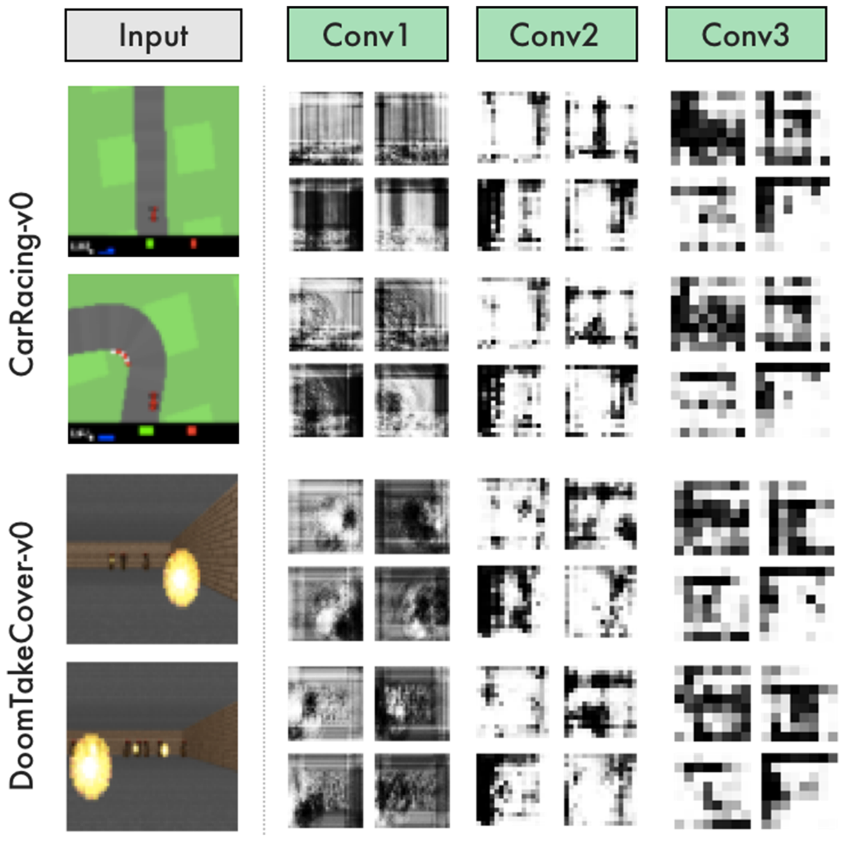


Fig. 4 Examples of the visual features in each convolution layer. The features in the same column are extracted by the same network

t-th features are $S(t) = \mathrm{CRC}_i(X(t))$, and w_i is the weight which transform multidimensional features that are decided by random seed i. S^i is a series of features extracted by CRC_i, we use the same environment state images and extract other features by feature extractors with different seed. If $\rho(i,j) \in [-1,1]$ is high, there is a strong linear relation between the features extracted by seed i and seed j.

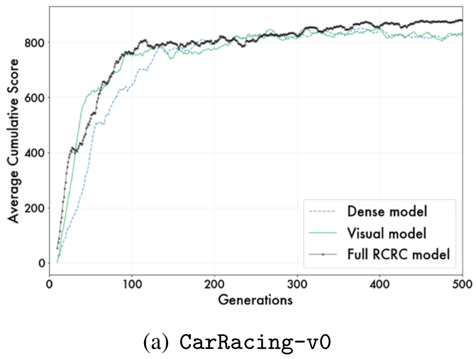
We collect the features 20 trials with random actions for `CarRacing-v0` and 100 trials with random actions for `DoomTakeCover-v0`. Finally, in `CarRacing-v0` $S^i \in \mathbb{R}^{(20000,1024)}$ and in `DoomTakeCover-v0` $S^i \in \mathbb{R}^{(21823,1024)}$ for different 20 random seeds. Both features are not including bias term. We calculate every pair (i.e. 190 pairs) of random seeds to calculate maximum canonical correlation.

As a result, the average maximum canonical correlation is 0.99947 ± 0.0003 in `CarRacing-v0` and that is 0.99959 ± 0.0004 in `DoomTakeCover-v0`. Therefore there is a strong linear relationship among the features extracted by other weights defined by different random seeds, and it can be concluded that the proposed method is robust against random seed choice in out settings.

5.2 CarRacing-v0

The best scores among 16 workers are shown in Fig. 5a. Each worker's score is evaluated as an average score over 8 randomly generated tracks. The dense model reached an average score above 880 over 8 randomly generated tracks, and the visual model reached above 890. The dense model's score transition has higher variance than the visual model's score transition. Furthermore, the visual model's score is less stable than the full RCRC model's score. These results shows that only one dense layer can extract visual features despite the fact that the weights are random and fixed, and the features extracted by the convolutional layers and the ESN improved scores.

The generalization ability of the visual model and the full RCRC model which evaluated as an average score over 100 random trials are shown in Table 1. The scores of comparison models are also shown in Table 1. The visual model which uses 512-dimensional visual features achieved 864 ± 79 which is better than the V model that uses 32-dimensional features extracted by VAE as input to controller layer in the world model. In addition, the full RCRC model reached 902 ± 21 which is comparable to state of the art approaches such as the world model approach [9] and genetic algorithm (GA) approach [33]. Therefore the full RCRC model can be regarded as having ability to solve `CarRacing-v0`. These results show the time-series features extracted by the ESN improves driving skill. Furthermore, mention about the difference between the visual model and the V model, the dimension of the extracted feature is different, and for this reason, it seems that



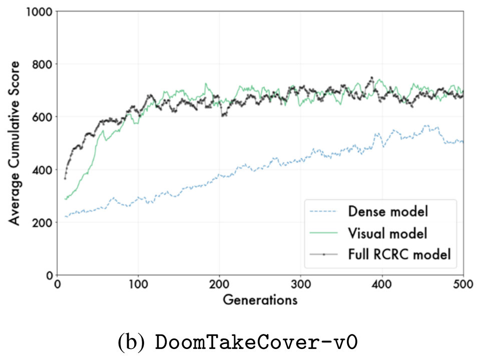


Fig. 5 **a** The best average score over 8 randomly created trials among 16 workers in `CarRacing-v0`. **b** The best average score over 8 randomly created trials among 32 workers in `DoomTakeCover-v0`. For simplicity, we plotted moving average of 10 scores

the difference of the score caused. In other words, it seems that enough visual features improves the score even it is extracted by random and untrained weight.

Compare to World model with random MDN-RNN [32] and GA approach [33], the RCRC model achieved a higher score. These approaches are similar to our approach. They use a fixed random-weight network to extract visual or time-series features, but we use the fixed random-weight network to extract both of them and achieved a higher score than those. This result shows that the visual and time-series features extracted by the RCRC model have enough information to solve the task.

Compare to the score of DQN approach [40] and DDQN with dropout approach [41], the RCRC model reached a higher score and solved the task. To mention about DQN, it estimates future rewards in each frame, but it tends to overestimate the rewards. If this happens in `CarRacing-v0`, it is likely that the car becomes fast to try to get a high score but becomes too fast to corner well. It seems the DQN approach couldn't assume a risk. On the other hand, Double DQN (DDQN) is the model that can estimates future rewards more severely than DQN. The scores of DDQN with dropout approach[41] is much better than that of DQN approach. This means that the overestimates of future rewards are fatal in solving `CarRacing-v0` and if the model could estimate future rewards properly, the model will perform well. Compare to these models, the RCRC model doesn't estimate future rewards. It only extracts visual and time-series features and use these features to optimize actions to maximize total rewards, so there is no risk of overestimates in the RCRC model.

Table 1 `CarRacing-v0` scores of various methods

Method	AVG. Score
DQN [40]	343 ± 18
DDQN with dropout [41]	892 ± 41
A3C (continuous) [42]	591 ± 45
A3C (discrete) [43]	652 ± 10
World model with random MDN-RNN [32]	870 ± 120
World model [9]	
V model	632 ± 251
World model	**906** ± 21
GA [33]	**903** ± 73
Weight Agnostic Neural Networks [44]	
Random weights	-69 ± 31
Random shared weight	375 ± 177
Tuned shared weight	608 ± 161
Tuned weights	893 ± 74
RCRC model (Visual model)	864 ± 79
RCRC model	**902** ± 21

Asynchronous Advantage Actor-Critic (A3C) is a model that train their model asynchronously in parallel environments. A3C uses Advantage method. The Advantage method uses the relative value of each action. It is evaluated by subtracting the value of the state from the value of each action. This makes model can estimates pure value of each action. Moreover, A3C also uses actor-critic method. The actor-critic method separates the training process of action decision-maker and value estimators. This method makes the training process more stable. A3C is similar to the RCRC model in that it uses less memory because it doesn't need to collect training data. Even our model has the advantages of A3C and the computation cost is low because of no need of gradient calculation, it reached a much higher score than that of A3C approaches [42, 43] and solve the task. Compare to those approaches, the RCRC model doesn't train feature extractor and doesn't train model in each parallel environments. From this result, it can be said that the RCRC model is not only having desirable characteristics but also a good performer.

The full RCRC model also achieved a higher score than the scores of Weight Agnostic Neural Networks (WANN) [44]. The WANN is a model that optimizes network architectures by no weight training. The WANN learn the network architectures with constraints that make it a simple architecture. To measure the generalization ability of the WANN, the following 4 weights are used in the evaluation: Random weights, Random shared weight, Tuned shared weight, and Tuned weights [44]. The Random weight means individual weights in the network which architecture is optimized, are drawn from uniform distribution $U(-2,2)$; The Random shared weight means all weights are same value which is drawn from $U(-2,2)$; The Tuned shared weight means using the highest performing shared weight value in range $(-2,2)$; The Tuned weights means individual weights are trained to solve task [44]. The concept of the WANN is a little similar to the RCRC in don't train weights, but different in what they focus on. The RCRC focuses on the random weights feature extractors ability and the WANN focuses on the simple network architectures that can solve tasks. Therefore, the WANN seems to have difficulty to achieve a high score with no weight training. These results show that to solve `CarRacing-v0`, it is important to extract meaningful or enough visual and time-series features.

5.3 DoomTakeCover-v0

The best scores among 32 workers are shown in Fig. 5b. Each worker's score is evaluated as an average score over 8 randomly generated trials. While the dense model only improved score little by little, the visual model and the full RCRC model improved scores fast and reached above 750 in early steps. Therefore, it seems that it is difficult to express complex visual features such as the depth of screens with the dense layer alone, and the convolutional layer is effective.

The generalization ability of the visual model and the full RCRC model which is evaluated as an average score over 100 random trials are shown in Table 2. For comparison, the scores of Random Policy Baseline that takes actions randomly, the OpenAI Gym leaderboard [31], and the self-playing world model with different temperature parameter τ are listed. The temperature τ controls the variance of the next environment state prediction. A large τ means that the model predicts next environment state with high variance. On the other hand, if τ sets to 0, the model predicts the next environment state deterministically.

The full RCRC model reached 922 ± 450 and the visual model achieved 832 ± 483. They couldn't reach the best score of the self-playing world model 1092 ± 556, but they greatly exceed above 750 which means "solved" the task. Although the visual model and the full RCRC model have similar score transition in the parameter optimization process, at average score over 100 random trials, the visual model's score is lower than the full RCRC model's one. It can be considered to be due to the fact that it might fall into a local optima easy, since the scores are high variance and each parameter is evaluated by only 8 trials during parameter optimization. Because the full RCRC model score has a higher score and lower variance than the visual model, it can be regarded as using time-series features extracted by the RCRC model are effective to solve tasks and can improve the generalization ability. This is assumed to caused by that the risk of falling into a local optima decrease because the model does not only depend on visual features by using time-series features.

While the full RCRC model reached a comparable score to the best score of the world model in `CarRacing-v0`, the full RCRC model couldn't reach the best score of the world model in `DoomTakeCover-v0`. The world model uses VAE and MDN-RNN, and can extract probabilistic features based on the assumption of multiple future environment states, but the RCRC model can only extract deterministic features by actual image input. In `DoomTakeCover-v0`, the environment state images are first-person view, and not all states can be observed. Therefore, it seems that the world model can get a higher score.

Furthermore, the RCRC model's feature extractor which is completely identical between tasks has generalization ability to solve both tasks by training only a linear transformation from the extracted features to the actions, despite the fact that the network's weights are set to random and fixed.

Table 2 `DoomTakeCover-v0` scores of various methods

Method	AVG. Score
Random Policy Baseline	210 ± 108
Gym Leader [31]	**820** ± 58
World model [9]	
$\tau = 0.10$	193 ± 58
$\tau = 0.50$	196 ± 50
$\tau = 1.00$	**868** ± 511
$\tau = 1.15$	**1092** ± 556
$\tau = 1.30$	**753** ± 139
RCRC model (Visual model)	**832** ± 483
RCRC model	**922** ± 450

6 Conclusions and discussions

In this study, we focused on extracting features that sufficiently express the environment state, rather than those that are trained to get higher rewards. To this end, we developed a novel approach called RCRC model which using fixed random-weight CNN and a novel ESN-based method, respectively, extracts visual features from environment state images and time-series features from transitions of visual features. This model architecture results in highly practical features that omit the training process of the feature extractor and reduce computational costs, and there is no need to store large volumes of data. Surprisingly, extracted features are expressive enough to solve multiple RL tasks with training only a linear transformation of those features, despite the fact that it used the completely identical feature extractor. These results bring us to the conclusion that network structures themselves, such as CNN and ESN, have the capacity to extract features, and the RCRC model has generalization ability to express various environments and solve RL tasks.

There are two main novelties in this paper. First, the RCRC is the first work as we know which combines the CNN and the reservoir computing method for task to extract features from transitions of images. Second, we showed the RCRC with an identical random-weight feature extractor can be used to multi-task and it can achieve comparable scores to state of the art approaches.

Although the RCRC model is not suitable for the tasks that are hard to simulate because it optimizes parameters by the simulated score with current parameters, it has the potential to make RL widely available. Recently, many RL models have achieved high performance in various tasks, but most of them have high computational costs and often require significant time for training. This makes the introduction of RL inaccessible to many people. However, by using the RCRC model anyone can build high-performance models fast with much lower computational costs. In addition, the RCRC model can handle a wide range of actions, and even when the environment changes, training can be performed without any pre-training. Therefore, the RCRC model can be used easily by anyone to apply to various environments.

The RCRC model can handle high-dimensional features such as video and image transitions without training feature extractors when rewards can be defined. Therefore, in real-world applications, the RCRC model has several strong points. It can be applied to tasks that have tremendous data and tasks which cannot apply batch predictions because the data distributions change dynamically. For example, we consider that it can be applied to the tasks that require dynamical high dimensional time-series features such as in-video advertisement optimization, automated driving system, and robot control. Note that how complex tasks the RCRC model can handle must be considered in future work.

As a further improvement, there is a possibility that the score can be improved by ensembling multiple features that are extracted by multiple convolutional reservoir computing layers as in the ESN [45]. In addition, generally reservoir computing is limited to tasks that do not require long-term memory because it is known that it has less memory capacity than LSTM, but the stacking ESN approach such as DeepESN [46] has possibility to improve memory capacity.

In future work, we consider making predictions from previous extracted features and actions to the next ones to be an important and promising task. Because the ESN was initially proposed to predict complex time-series, it can be assumed to have capacity to predict next features. If this task is achieved, it can self-simulate RL tasks by making iterative predictions from an initial state. This will help to broaden the scope of RL applications.

Acknowledgements The authors are grateful to Takuya Yaguchi for the discussions on reinforcement learning. We also thank Hiroyasu Ando for helping us to improve the manuscript.

Author Contributions Both authors contributed to the study conception, design, coding, analysis and trial experiments. The submitted experimental results were performed by Hanten Chang. The first draft of the manuscript was written by both authors and both authors read and approved the final manuscript.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

1. 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
2. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller MA (2013) Playing atari with deep reinforcement learning. arXiv:1312.5602
3. Horgan D, Quan J, Budden D, Barth-Maron G, Hessel M, van Hasselt H, Silver D (2018) Distributed prioritized experience replay. arXiv:1803.00933
4. Kapturowski S, Ostrovski G, Dabney W, Quan J, Munos R (2019) Recurrent experience replay in distributed reinforcement learning. In: International conference on learning representations
5. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine 34(6):26–38
6. Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: 2015 AAAI Fall symposium series
7. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, pp 1928–1937
8. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv:1511.05952
9. Ha D, Schmidhuber J (2018) Recurrent world models facilitate policy evolution. In: Advances in neural information processing systems. Curran Associates Inc., pp 2450–2462
10. Kingma DP, Welling M (2013) Auto-encoding variational bayes. arXiv:1312.6114
11. Rezende DJ, Mohamed S, Wierstra D (2014) Stochastic backpropagation and approximate inference in deep generative models. arXiv:1401.4082
12. Graves A (2013) Generating sequences with recurrent neural networks. arXiv:1308.0850
13. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
14. Hansen N (2016) The CMA evolution strategy: A tutorial. arXiv:1604.00772
15. Lukoševičius M, Jaeger H (2009) Reservoir computing approaches to recurrent neural network training. Comput Sci Rev 3(3):127–149
16. Jaeger H (2001) The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report 148 (34):13
17. Jaeger H, Haas H (2004) Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304(5667):78–80
18. Tanisaro P, Heidemann G (2016) Time series classification using time warping invariant echo state networks. In: 2016 15th IEEE

international conference on machine learning and applications (ICMLA). IEEE, pp 831–836
19. Ma Q, Shen L, Chen W, Wang J, Wei J, Yu Z (2016) Functional echo state network for time series classification. Inf Sci 373:1–20
20. Szita I, Gyenes V, Lőrincz A (2006) Reinforcement learning with echo state networks. In: International conference on artificial neural networks. Springer, pp 830–839
21. Bush K, Anderson C (July 2005) Modeling reward functions for incomplete state representations via echo state networks. In: Proceedings. 2005 IEEE international joint conference on neural networks, 2005, vol 5, pp 2995–3000
22. Chang H-H, Song H, Yi Y, Zhang J, He H, Liu L (2018) Distributive dynamic spectrum access through deep reinforcement learning: A reservoir computing-based approach. IEEE Internet of Things Journal 6 (2):1938–1948
23. Tong Z, Tanaka G (2018) Reservoir computing with untrained convolutional neural networks for image recognition. In: 2018 24Th international conference on pattern recognition (ICPR). IEEE, pp 1289–1294
24. Lukoševičius M (2012) A practical guide to applying echo state networks. In: Neural networks: Tricks of the trade. Springer, pp 659–686
25. Inubushi M, Yoshimura K (2017) Reservoir computing beyond memory-nonlinearity trade-off. Sci Rep 7 (1):10199
26. Chang H, Nakaoka S, Ando H (2019) Effect of shapes of activation functions on predictability in the echo state network. arXiv:1905.09419
27. Verstraeten D, Schrauwen B, d'Haene M, Stroobandt D (2007) An experimental unification of reservoir computing methods. Neural Networks 20(3):391–403
28. Goudarzi A, Banda P, Lakin MR, Teuscher C, Stefanovic D (2014) A comparative study of reservoir computing for temporal signal processing. arXiv:1401.2224
29. Klimov O (2016) Carracing-v0 https://gym.openai.com/envs/CarRacing-v0/
30. Kempka M, Wydmuch M, Runc G, Toczek J, Jaśkowski W (2016) ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In: IEEE conference on computational intelligence and games. The best paper award. IEEE, Santorini, pp 341–348
31. Paquette P (2016) Doomtakecover-v0 https://gym.openai.com/envs/DoomTakeCover-v0/
32. Tallec C, Blier L, Kalainathan D (2018) Reproducing "world models". is training the recurrent network really needed ? https://ctallec.github.io/world-models/
33. Risi S, Stanley KO (2019) Deep neuroevolution of recurrent and discrete world models. In: Proceedings of the genetic and evolutionary computation conference, GECCO '19. ACM, New York, pp 456–462
34. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780
35. LeCun Y (1998) The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/
36. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv:1606.01540
37. Wydmuch M, Kempka M, Jaśkowski W (2018) Vizdoom competitions: Playing doom from pixels. IEEE Transactions on Games
38. Lukosevicius M (2012) A practical guide to applying echo state networks. In: Neural networks: Tricks of the trade
39. Hardoon DR, Szedmak S, Shawe-Taylor J (2004) Canonical correlation analysis: An overview with application to learning methods. Neural Computation 16(12):2639–2664
40. Prieur L (2017) Deep-q learning for box2d racecar rl problem
41. Gerber P, Guan J, Nunez E, Phamdo K, Monsoor T, Malaya N (2018) Solving openai's car racing environment with deep reinforcement learning and dropout https://github.com/AMD-RIPS/RL-2018/blob/master/documents/nips/nips_2018.pdf
42. Se WJ, Min J, Lee C (2017) Reinforcement car racing with a3c. https://www.scribd.com/document/358019044/
43. Khan M, Elibol OH (2018) Car racing using reinforcement learning. https://web.stanford.edu/class/cs221/2017/restricted/p-final/elibol/final.pdf
44. Gaier A, Ha D (2019) Weight agnostic neural networks. In: Wallach H, Larochelle H, Beygelzimer A, d' Alché-Buc F, Fox E, Garnett R (eds) Advances in Neural Information Processing Systems, vol 32. Curran Associates, Inc., pp 5365–5379
45. Massar M, Massar S (2013) Mean-field theory of echo state networks. Physical Review E 87(4):042809
46. Gallicchio C, Micheli A, Pedrelli L (2017) Deep reservoir computing: a critical experimental analysis. Neurocomputing 268:87–99