Improve PID controller through reinforcement learning

Conference Paper · August 2018

DOI: 10.1109/GNCC42960.2018.9019095

4 authors, including:

Weiguo Zhang
Northwest A & F University
87 PUBLICATIONS **211** CITATIONS
SEE PROFILE

Liu Jinglong
Shanghai Institute of Aerospace System Engineering
12 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Improve PID controller through reinforcement learning

Yunxiao Qin, Weiguo Zhang, Jingping Shi, Jinglong Liu.

***Abstract*—Deep Reinforcement Learning (DRL) algorithm provides an effective and powerful way for the computer learning to make decisions according to what the computer sees, the most famous application of DRL is training the computer to play computer games, such as the computer Go and Atari Games. In this paper, we present that the DRL can used to improve the classical PID controller by training an adaptive PID controller with its parameters change according to the changing state, we name our new controller as DRPID, the deep reinforcement learning algorithm that we used to train the PID controller in this paper is deep deterministic policy gradient, which is very suitable to solve continue action control problem. We test our DRPID algorithm by training it to control an inverted pendulum in the OpenAI gym simulation environment, and we found DRPID controller works very well, outperforms the common PID controller with fixed parameters, by a great margin.**

I. Introduction

The classical Proportional Integral and Derivative (PID) control algorithm is an effective and simple control algorithm in the auto control field[1]， it's structure is show in figure 1. Even nowadays, PID is still the most popular control algorithm used in industry, more than 80% controller in industry use PID or its variants to solve auto control problems[1], [2].

In the procedure of PID controller application, tuning the parameters of PID k_p, k_i, k_d for a stable, fast and accuracy control performance is an usual procedure[3], these parameters will be fixed after tuning procedure. The PID controller with fixed parameters is very suitable to solve linear control process, However, in practice, the time varying system and nonlinear system is more common, the PID controller may work well when the control object stay in the vicinity of balance point, but when the control object goes far away from the balance point, the PID controller will not perform well. For solving this problem, PID parameters should be adaptive according to the state of the control object.

Furthermore, the tuned parameters is usually not the optimal, as the tuning procedure depend on engineer's experience. In this paper we try to solve the problems of classical PID by integrate it with deep reinforcement learning method.

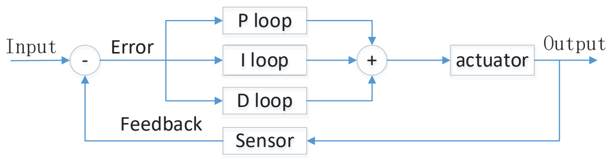

Figure 1. The structure of a classical PID controller

Recently, with the rapid develop of machine learning algorithm, mostly the deep learning[4] and reinforcement learning[5] algorithms, there are several great breakthroughs in Artificial Intelligence, such as the AlphaGo[6], [7]， designed by Google DeepMind, is the first computer Go program that can defeat the most skillful human players, and the program based on deep reinforcement learning algorithm can play Atari games better than human players[8]. In this paper, we design an algorithm named with DRPID that combine common PID algorithm with deep reinforcement learning, outperforms the classical PID algorithm very much.

The basic idea behind our algorithm is using DRL to train an agent to output the parameters of PID according to current state, as show in figure 2. When the agent output suitable parameters, the PID controller will control the actor well, and it will receive a higher reward as encouragement, otherwise, a lower reward will be received as punishment, as the agent continue be trained, the parameters output by the agent will converge to the optimal value gradually. In this paper, we use deep deterministic policy gradient algorithm to train the agent.

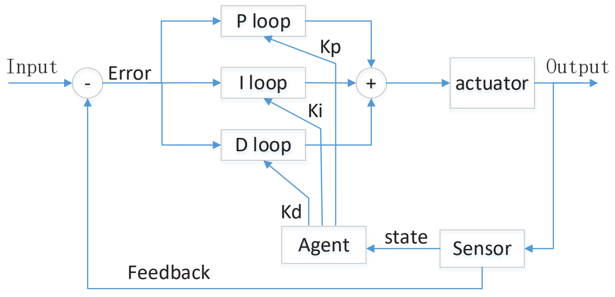

Figure 2. The structure of a DRPID controller

*Research supported by National Natural Science Foundation of China under grant #61573286 and National Defense 973 Program under grant #6131890302.

Yunxiao Qin is with the Flight control and simulation Laboratory, Northwestern Polytechnical University, Xi'an, Shaanxi, 710129, China. (E-mail: qyxqyx@mail.nwpu.edu.cn).
Weiguo Zhang is with the Flight control and simulation Laboratory, Northwestern Polytechnical University, Xi'an, Shaanxi, 710129, China. (E-mail: zhangwg@nwpu.edu.cn).
Jingping Shi is with the Flight control and simulation Laboratory, Northwestern Polytechnical University, Xi'an, Shaanxi, 710129, China. (E-mail: shijingping@nwpu.edu.cn).
Jinglong Liu is with the Flight control and simulation Laboratory, Northwestern Polytechnical University, Xi'an, Shaanxi, 710129, China. (Phone: +86-13474160763; E-mail: ljlong@mail.nwpu.edu.cn).

II. BACKGROUND

A. PID control algorithm

Consider problems of the classical PID controller, there are many researches proposed different varieties of PID controller which combine classical PID control and other control techniques, especially the intelligent control techniques. For example, using fuzzy neural network[9] to tune a PID controller is proved can improve system effectiveness and robustness. Another example is supervise the PID controller by an Adaptive Neuro-Fuzzy Inference System (ANFIS)[10]. In the experiment of control a brushless DC motor, it is shown that the PID controller supervised by ANFIS outperformed other controllers.

The performance of classical PID controller can also be improved by using adaptive methodologies, when the controller was applied to nonlinear system, the simulation in [11] shows the adaptive method can reach a better performance than the classic PID controller.

B. deep reinforcement learning

The most important goal of artificial intelligence (AI) is to build fully autonomous agent that can learn and act like human, recent years, with the rapid development of deep reinforcement learning[12], there are several great breakthroughs in the AI field, and the DRL becomes the most promising technique to achieve the goal of AI.

The DRL technique is built by combining the classical reinforcement learning algorithm and the recent developed deep learning algorithm.

The classical reinforcement learning algorithm is an experience-driven algorithm. There are three important elements in reinforcement learning: environment, agent and reward. The agent learns by interacting with the environment, at each time step, the agent make a decision of what to do according to state or observation of the environment, and then the environment will change according to the agent's action, at the same time, the agent will get a reward, the reward is a real number, and the goal of the agent is to maximize the sum of all the rewards in one episode. By interact with the environment constantly, the agent will learn what action will get higher reward at given state or observation, and what action will cause lower reward. The reinforcement learning algorithm is powerful, but it has several problems, such as huge memory cost when the dim of state and action increase, and computational complexity. Thanks to the rapid development of deep learning and the computer capacity, the reinforcement learning can solve its problems by combine itself with deep learning, then it get a new name: deep reinforcement learning.

Deep learning is a very powerful machine learning algorithm based on back-propagation neural network[13], it is a data-driven algorithm. Benefit from the elevation of computer capacity, the growth of data all over the world, the problem of data hunger and computation complexity have been relieved greatly, the improvement of active functions, the optimization of neural network structure(ResNet[14], DenseNet[15] and so on), the dropout[16] method, batch normalization[17] and so on, the main problems of deep learning (data hunger, gradient vanish or explosion, computation complexity and overfitting) have been solved almostly.

With so many successful techniques to strengthen the deep learning algorithm, the deep learning algorithm shows overwhelming advantages towards other machine learning algorithms, especially in the field of computer vision[18]–[20], nature language processing and speech recognition, nowadays, deep learning is spreading very fast to many industries.

By combining the powerful deep learning algorithm and reinforcement learning algorithm, we get Deep reinforcement learning (DRL) algorithm which is the core technique of the most skillful and famous computer Go program: AlphaGo[6], [7].

The first success DRL algorithm is DQN[8], it is a combination of Q-learning and deep learning, by replace the tabular of Q value with deep learning model, this algorithm shows great performance in playing Atari games, it was showed that the agent can play better than human players in many games.

After DQN, there are several algorithms that improve DQN, such as the Double DQN[21] solve the problem of overestimate of DQN, the Dueling DQN[22] separate the Q value into two part: state value and action value, improve the DQN greatly.

The algorithms of DQN series are value base algorithms, means the agent choose actions according to the Q value of state action pair, the action which makes the largest Q-value will be choose. The value based method is not capable to solve the continue action control problem, so in this paper, we use deep deterministic policy gradient(DDPG)[23]，an algorithm which developed on policy gradient[24] and actor-critic[25]. DDPG combine ideas of both the value based and policy based algorithms, output actions deterministically[26], can solve the continue action control problem very well.

III. ALGORITHM

In this paper, we use DDPG reinforcement learning algorithm to train the agent to output the parameters of the PID controller according to the state of the control object. In the original reinforcement learning algorithm, the process of the agent output actions is a Markov decision process (MDP), means the agent make action decision only depend on the current state, for the deterministic decision making case, the decision making process can be written in equation (1).

$$a_k = \pi(s_k) \tag{1}$$

In this paper, what the agent output is the parameters of the PID controller, show as equation (2):

$$(k_{kp}, k_{ki}, k_{kd}) = a_k = \pi(s_k) \tag{2}$$

For the agent can output more better parameters, we modify the DDPG algorithm slightly, the agent make decisions not only depend on the current state s_k, but also the past L steps state, as the equation (3) shows.

$$(k_{kp}, k_{ki}, k_{kd}) = a_k = \pi(s_k, s_{k-1}, \cdots, s_{k-L}) \tag{3}$$

The discrete formation of PID control algorithm is show as equation (4). It is obvious that the output of PID control is not only depend on the current error, but also depend on the passed error, especially the integral loop sums all the passed errors.

$$u(k) = k_p * error(k) + k_i * \sum_{i=1}^{k} error(i) + k_d * (error(k) - error(k-1)) \tag{4}$$

The error is calculated by equation (5)

$$error(k) = input(k) - feedback(k) \tag{5}$$

In this paper, we modify the integral loop of PID controller to sum only the current error and the passed L step errors. Then the PID controller can be modified to the formation of equation (6).

$$u(k) = k_p * error(k) + k_i * \sum_{i=k-L}^{k} error(i) + k_d * (error(k) - error(k-1)) \tag{6}$$

The total learning and control procedure of DRPID is shown as:

```
1. Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor
   $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
2. Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$,
   $\theta^{\mu'} \leftarrow \theta^\mu$
3. Initialize replay memory buffer R
4. episode = 1
5. Initialize a random process $N$ for exploration
6. Receive initial observation state $s_1$
7. t = 1
8. create the sequence: $s^t_{sequence} = [s_t, s_{t-1}, \cdots, s_{t-L}]$, (the $s_{t-i}$
   can be filled by 0 vector if $t-i<1$)
9. get the parameters of PID controller
   $(K_{pt}, K_{it}, K_{dt}) = p_t = \mu(s^t_{sequence} | \theta^\mu) + N_t$
10. calculate the control signals by PID function:
    $u_t = PID(p_t, s^t_{sequence})$
11. input the control signal to the simulation environment,
    observe reward $r_t$ and new state $s_{t+1}$
12. create the new sequence: $s^{t+1}_{sequence} = [s_{t+1}, s_t, \cdots, s_{t-L+1}]$
13. Store transition ($s^t_{sequence}, p_t, r_t, s^{t+1}_{sequence}$) in R
14. Sample a random mini batch of N transitions
    ($s^t_{sequence}, a_t, r_t, s^{t+1}_{sequence}$) from R to train both the critic
    and actor network of DDPG
15. Update the target networks:
        $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
16. $s^t_{sequence} = s^{t+1}_{sequence}$
17. If t=T or the simulation environment return 'done', jump
    to step 18, else t=t+1, and jump to step 9
18. If episode<M, then episode=episode+1 and jump to step 6,
    else store the networks' weight of DDPG, and the
    program is end
```

IV. EXPERIMENT

A. *Introduction of the simulation environment*

In this paper, we test our algorithm in the inverted pendulum simulation environment of the OpenAI gym module, as show in figure3. In the inverted pendulum simulation environment, the state of the inverted pendulum contain four elements: x, derivation of x, theta, derivation of theta. The control signal input to the inverted pendulum is two number: 1 or -1, means slide the block right or left.

In every episode, the inverted pendulum is initialized, the four elements of the state is set to random value in range of -0.05 to 0.05, the reward of each control step is defined as: the reward will be set to 1 if the state of the inverted pendulum satisfy the requirement: the absolute value of x smaller than $x_{threshold}$ (default is 2.4 meter) and the absolute value of theta smaller than $\theta_{threshold}$ (default is 0.2 radian), otherwise the reward will be set to 0. Whenever inverted pendulum goes against the requirement above, the simulation environment will return a 'done' signal, means current episode is over. For preventing any episode be endless, we should define a maximum control steps of one episode marked T, if the inverted pendulum has been controlled by T times in one episode, the episode is over, no matter have the simulation environment return the 'done' signal. The performance of the control algorithms or the deep reinforcement learning algorithms can be represented by the sum of all the rewards in one episode.

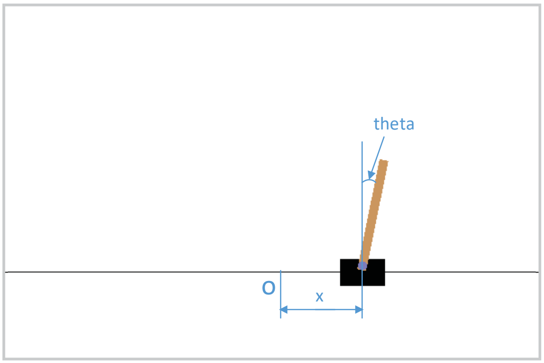

Figure 3 the inverted pendulum simulation environment of the OpenAI gym module, there is a pendulum on the sliding block, and we should control the sliding block to slide left or right for preventing the pendulum fall down. O is the base point of the horizontal axis, and the coordinate of the slide block is x, and the angel between the pendulum and the vertical direction is theta.

In this paper, we modify the simulation environment for it be more suitable for our experiment, what we modified are:

1. Considering the output of PID controller is continue signal, we modify the simulation environment for it can accept continue signals range from -1 to 1.
2. We change the reward function to equation (7), the $abs(x)$ is the function that calculate the absolute value of x the parameter p and q is used to guild the inverted pendulum's behavior, if parameter p is set be a large value, then the agent will pay more attention to control inverted pendulum will be closed to the base point O, if we set q be a large value, then the agent will pay more attention to control inverted pendulum be vertical.

$$reward = p*(1-abs(x)) + q*(0.1-abs(theta)) \tag{7}$$

B. Transition sample memory

The transition sample memory is a memory used to store the latest M=100000 transition samples, the samples will be used as training data to train the agent. Each transition is formed as: { $s^t_{sequence} = [s_t, s_{t-1}, \cdots, s_{t-L}]$, $(K_{pt}, K_{it}, K_{dt}) = p_t$, r_t, $s^{t+1}_{sequence} = [s_{t+1}, s_t, \cdots, s_{t-L+1}]$ }, means the agent output the parameters of p_t according to $s^t_{sequence}$, then we output control signal by PID, and the environment turn to next state, with a reward return to the agent. In the beginning, the memory is initialized to empty, the transition samples is sampled one by one from the simulation environment, and written to the memory.

C. neural networks in DDPG

The structure of actor neural network in DDPG is show in figure 4(a), we input the state sequence $s_k, s_{k-1}, \cdots, s_{k-L}$ to the actor network, output the parameters: k_p, k_i, k_d. The structure of critic neural network in DDPG is show in figure 4(b), we input both the state sequence $s_k, s_{k-1}, \cdots, s_{k-L}$ and the parameters k_p, k_i, k_d to the critic network, output the value of the pair: $v((s_k, \cdots, s_{k-L}), (k_p, k_i, k_d))$

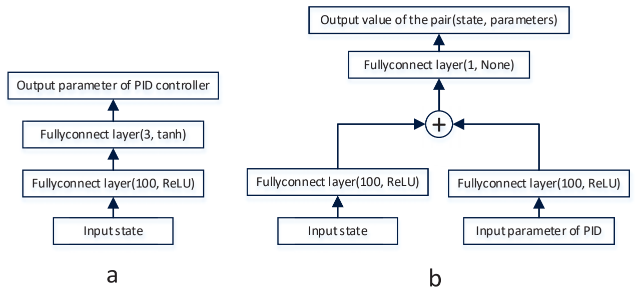
Output parameter of PID controller
Fullyconnect layer(3, tanh)
Fullyconnect layer(100, ReLU)
Input state
a
Output value of the pair(state, parameters)
Fullyconnect layer(1, None)
+
Fullyconnect layer(100, ReLU)
Fullyconnect layer(100, ReLU)
Input state
Input parameter of PID
b

Figure 4 The structure of actor(left) and critic(right) neural network in DDPG.

D. Training

After the memory is full, the training process is startup, we randomly read Batch-size=5120 transition samples from the memory to train the neural network in DDPG. The optimization algorithm is Adam with the learning rate of 0.001 for actor and 0.002 for critic respectively at beginning, and both the learning rate will decay exponentially. We use soft parameter update method with τ =0.01 to update the parameter of actor-target network and critic-target network. There are 6000 episodes for the DRPID can be trained thoroughly, and the maximum steps in one episode T is set to 5000. The parameters p and q in equation (7) were set to 1 and 10 respectively.

E. Compare performance of different controllers

For comparing the original PID controller and the DRPID controller, we use the original PID controller with hand adjusted parameters to control the inverted pendulum to be vertical, in our experiment, the parameters is: 0.7, 0.001, 0.3, the PID controller with these parameters performs very well in the inverted pendulum simulation environment.

Because we have modified the reward function of the inverted pendulum simulation environment, so we can only compare performance of different algorithms by compare the number of control steps in one episode of these algorithms respectively, the controller which can hold the inverted pendulum with more steps in one episode is better.

Considering some reader may wondering why not control the inverted pendulum by DDPG directly, we do experiment to test the performance of direct DDPG control. In the direct DDPG control experiment, we use the same hype parameters as the DRPID, the difference between these two algorithm is that the direct DDPG controller output control signals directly. Besides, we tested another controller: DRPD(an controller abandon the integral loop from DRPID), with the same hype parameters with DRPID too.

The performance comparison between original PID, DRPID, DRPD, DDPG control algorithms is show in figure 5

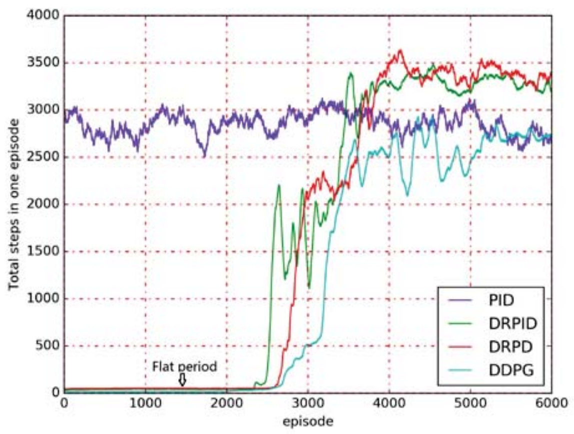

Figure 5. The four color lines represent the performance of original PID, DRPID, DRPD, DDPG controller. There is a flat period in DRPID, DRPD, DDPG controllers, because these three controllers will collect enough control experience (transitions samples) at the beginning episodes, and the controller will learn from these collected experience only when the experience memory is full.

From figure 5, we can see, after about 4500 episodes, all the DRPD, DRPID and DDPG controllers have been trained thoroughly. The average total control steps in one episode from episode 5000 to episode 6000 of PID, DRPID, DRPD and DDPG controllers are 2765.9, 3298.5, 3383.6 and 2698.2 respectively. the performance comparison relationship of the four controllers is： DRPD > DRPID > PID > DDPG, suggesting combine the DDPG algorithm with PID algorithm can outperform both of them.

It is a little interesting that we found DRPD controller outperforms the DRPID controller, the suitable reason we guess is that the DRPD controller with adaptive parameter k_p and k_d is sufficient to control the inverted pendulum well, add the integral loop in the controller may bring unnecessary control signals and cause overfitting.

We sample some parameters k_p , k_i and k_d in the DRPID controller, and they ware show in Table1, it is obvious that the parameters changes when the state change, and in some state, the parameters of the DRPID is closed with the parameters we turned in classical PID controller (0.7,0.001,0.3).

Current state				**parameters**		
x	v	θ	$\triangle\theta$	k_p	k_i	k_d
-1.05	**-0.087**	**-0.11**	**-0.194**	**1.26**	**0.002**	**0.31**
-1.04	**-1.23**	**-0.10**	**0.48**	**1.17**	**-0.003**	**0.29**
-0.08	**-0.18**	**-0.004**	**0.127**	**0.61**	**0.02**	**0.6**
0.01	**-0.20**	**-0.01**	**-0.146**	**0.67**	**0.013**	**0.43**
1.06	**1.24**	**0.11**	**-0.138**	**0.98**	**0.001**	**0.36**
1.58	**1.41**	**0.16**	**0.324**	**1.32**	**-0.002**	**0.26**

TABLE I. THE PARAMETERS IN DRPID

V. CONCLUSION AND FUTURE WORK

From the experiment result, we can get conclusion that the classical PID controller can be improved very much by integrate it with DRL algorithms. However, everything has two faces, there is a problem of DRPID, because the DRPID need to be trained, and at the beginning of training, the performance of DRPID is too poor to be implemented in real world. For this problem, we have some ideas to solve it, for example, we can use the classical PID controller to train a weak DRPID controller first by imitation learning process, the weak DRPID controller will imitate the behavior (control signal) of classical PID controller like a student, after imitation learning, the week DRPID will perform not bad, than we can use it to control the actuator directly, and the week DRPID will be trained with reinforcement learning algorithm simultaneously, finally, the DRPID will outperform PID gradually and become a strong DRPID controller.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under grant #61573286. This work is also supported by the National Defense 973 Program under grant #6131890302.

REFERENCES

[1] K. J. Åström and T. Hägglund, “The future of PID control,” *Control Eng. Pract.*, vol. 9, no. 11, pp. 1163–1175, Nov. 2001.

[2] J. G. Ziegler and N. B. Nichols, “Optimum settings for automatic controllers,” *InTech*, vol. 42, no. 6, 1995.

[3] J. C. Shen, “Fuzzy neural networks for tuning PID controller for plants with underdamped responses,” *Fuzzy Syst. IEEE Trans. On*, vol. 9, no. 2, pp. 333–342, 2001.

[4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, 2015.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “A Brief Survey of Deep Reinforcement Learning,” *ArXiv170805866 Cs Stat*, Aug. 2017.

[6] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[7] D. Silver *et al.*, *Mastering the game of Go without human knowledge*, vol. 550. 2017.

[8] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” *ArXiv13125602 Cs*, Dec. 2013.

[9] S. J. Ho, L.-S. Shu, and S.-Y. Ho, “Optimizing fuzzy neural networks for tuning PID controllers using an orthogonal simulated annealing algorithm OSA,” *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 3, pp. 421–434, Jun. 2006.

[10] K. Premkumar and B. V. Manikandan, “Fuzzy PID supervised online ANFIS based speed controller for brushless dc motor,” *Neurocomputing*, vol. 157, pp. 76–90, 2015.

[11] P. K. Kolavennu, S. Palanki, D. A. Cartes, and J. C. Telotte, “Adaptive controller for tracking power profile in a fuel cell powered automobile,” *J. Process Control*, vol. 18, no. 6, pp. 558–567, 2008.

[12] Y. Li, “Deep Reinforcement Learning: An Overview,” 2017.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, p. 323533a0, Oct. 1986.

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *ArXiv E-Prints*, vol. 1512, p. arXiv:1512.03385, Dec. 2015.

[15] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely Connected Convolutional Networks,” *ArXiv E-Prints*, vol. 1608, p. arXiv:1608.06993, Aug. 2016.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[17] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *ArXiv E-Prints*, vol. 1502, p. arXiv:1502.03167, Feb. 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *ArXiv E-Prints*, vol. 1602, p. arXiv:1602.07261, Feb. 2016.

[20] S. Targ, D. Almeida, and K. Lyman, “Resnet in Resnet: Generalizing Residual Architectures,” *ArXiv160308029 Cs Stat*, Mar. 2016.

[21] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning,” *ArXiv150906461 Cs*, Sep. 2015.

[22] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” *ArXiv151106581 Cs*, Nov. 2015.

[23] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *ArXiv150902971 Cs Stat*, Sep. 2015.

[24] S. Kakade, “A natural policy gradient,” in *International Conference on Neural Information Processing Systems: Natural and Synthetic*, 2001, pp. 1531–1538.

[25] J. Peters and S. Schaal, “Natural Actor-Critic,” *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, Mar. 2008.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International Conference on International Conference on Machine Learning*, 2014, pp. 387–395.