Deep Reinforcement Learning with Regularized Convolutional Neural Fitted Q Iteration

Cosmo Harrigan

University of Washington

Abstract. We review the deep reinforcement learning setting, in which an agent receiving high-dimensional input from an environment learns a control policy without supervision using multilayer neural networks. We then extend the Neural Fitted Q Iteration value-based reinforcement learning algorithm (Riedmiller et al) by introducing a novel variation which we call *Regularized Convolutional Neural Fitted Q Iteration* (RC-NFQ) that incorporates convolutional neural networks similarly to the Deep Q Network algorithm (Mnih et al) and dropout regularization to improve generalization performance. Finally, we present an implementation of the algorithm and discuss several extensions.

1 Neural Networks

1.1 Introduction

Neural networks are non-linear parametric statistical models. There are multiple supervised and unsupervised learning algorithms associated with them. Learning multilayer neural network models, also known as *deep learning*, involves choosing a model type and an architecture for the model, and then learning suitable model parameters. While all such models consist of directed graphs of artificial neurons, many specific higher-level regularities have been proposed as building blocks, such as convolutional neural networks and recurrent neural networks. They are proposed in order to counteract the combinatorial explosion of possible network configurations based on the following observations about the domains operated upon in the natural world:

- The environment contains many statistical regularities
- Similar patterns tend to appear in multiple regions of space and time
- Natural scenes have a hierarchical, compositional structure

The most common learning algorithm for neural networks is the backpropagation algorithm [33, 21], which is a supervised learning method that uses labeled examples, and computes the difference between the network's output and the target output, and iteratively makes small adjustments to the network until its predictions better match the labeled examples.

An excellent survey of the field is available in [38] containing an overview of each subfield, along with historical developments and 888 references. A textbook, *Deep Learning* [3], is also available.

1.2 Convolutional Neural Networks

Convolutional neural networks [21, 20, 9, 3] are multi-layer perceptrons with particular constraints on their weights. They have demonstrated significant successes in recent years in the field of computer vision [34]. They demonstrate some similarities to the human visual cortex, although they also exhibit many differences [14, 1].

Within each layer there exists a set of *feature detectors*, each of which responds to the presence of a particular pattern in an input tensor. Each feature detector is applied at multiple locations in the input tensor. This application of identical feature detectors across the input is referred to as *weight sharing*. At each location, the extent to which the feature is present is calculated by the *discrete convolution* operator. To represent the information more compactly, *dimensionality reduction* techniques which aim to extract the most relevant components are utilized between layers; usually *max pooling* layers [21], which only consider the most prominent feature within a particular location, or, more recently, *strided convolutions* [42], which spatially separate the application of the feature detectors in such a way as to reduce the dimensionality of the output by reducing the amount by which they overlap.

1.3 Gradient Descent Algorithms

Gradient descent in neural networks is an optimization method that aims to minimize an objective function $J(\theta)$ parameterized by the parameter vector θ of a neural network, by iteratively updating a candidate solution based on the local gradient $\nabla_\theta J(\theta)$ of the function to be optimized with regards to its parameters, evaluated on some training examples. The solution is updated by taking steps in the opposite direction of the gradient. The magnitude of each update is controlled by a learning rate η.

In full-batch gradient descent, all of the training examples are used to compute the gradient at each step. A variant, called stochastic gradient descent, uses a single training example at each step. There is yet another variant, called mini-batch gradient descent, which lies between these two concepts and is the most commonly used. It uses a subset of the training examples at each step to compute the gradient, which can lead to faster convergence behavior.

There exist more sophisticated gradient descent algorithms that aim to adaptively change the learning rate on a per parameter basis as the optimization proceeds. One of these algorithms, which we will use in this work, is RMSprop [45] which is a mini-batch varient of the Rprop [32] batch gradient descent algorithm. Other common alternatives are Adagrad [7], Adadelta [50] and Adam [17].

2 Reinforcement Learning

2.1 Introduction

Reinforcement learning [44, 23] is a general framework for the problem of an *agent* embedded in an unknown *environment* that learns a behavior *policy* that maximizes a *reward function*. The standard reference for the field is [44], and a survey of recent techniques is found in [48]. Under the reinforcement learning paradigm, an agent sends actions to an environment, and receives observations and rewards in response.

Reinforcement learning belongs to the fields of machine learning and artificial intelligence. For a survey of definitions of artificial intelligence, see [23]. For a broad view of artificial intelligence in general, see [35]. For a universal theory of optimal artificial intelligence and reinforcement learning, refer to [15] and [22].

As a general framework, there are many specific methods that can be described within the reinforcement learning setting. Unsupervised learning methods can also be considered part of reinforcement learning when they are employed by the agent to improve its ability to achieve rewards. For a survey of unsupervised learning applied in this context, see [38, Section 6.4].

The most common formalism used to describe this setting involves the agent observing a state s and taking an action a, and the environment responding with a reward signal r along with a subsequent state s'. This agent-environment interaction is then repeated through time, defining a sequence of experience tuples (s, a, r, s') which can be used as input to various reinforcement learning algorithms.

If the environment is *fully observable*, then the observation at a particular time corresponds to the exact configuration of the environment; in more complex problems, the environment may only be *partially observable*, in which case the agent only has access to a particular subset of the true state of the environment through its observations.

The policy of an agent represents how the agent behaves, conditional on its observations of the environment and the state of its internal memory. There are many ways that a policy can be represented. It can be as simple as a lookup table, which is equivalent to a *reflex agent* [35] in artificial intelligence. Alternatively, the agent can parameterize its policy in some way, so that its responses become a more complex function of the state of the environment and its internal state.

A further distinction is made between *model-based* reinforcement learning, in which an agent learns an explicit model of the conditional *transition probabilities* $P(s'|s, a)$ of the environment, and *model-free* reinforcement learning, in which an agent does not explicitly model the environment. There are two major approaches within the model-free reinforcement learning setting: *direct policy search* methods and *value-based methods*, both of which we review below.

2.2 Exploration and Exploitation

An important tradeoff in reinforcement learning exists between the exploitation of current estimates to increase immediate reward and further exploration of the environment to improve the likelihood of future reward through improved accuracy. This is commonly referred to as the *exploration versus exploitation* tradeoff [44, Chapter 1].

2.3 Function Approximation

The simplest category of reinforcement learning problems are the *tabular* case, in which each state is explicitly defined, and state-specific values are learned. This corresponds to an explicit representation in the form of a lookup table with an entry for every single state. In all but the simplest domains, this approach is not practically feasible, due to the size of the state space.

As a result, *function approximators* are used to generalize from specific states into more abstract state representations. The concept of function approximation is related to the concepts of regularization and generalization in statistics, and compression in information theory. In general, a function approximator takes as input a high-dimensional state, and maps it to a lower dimensional representation. Both linear and non-linear function approximation methods are studied. Neural networks are a particular instance of non-linear function approximators.

2.4 Direct Policy Search

In *direct policy search*, the space of possible policies is searched directly. The agent does not attempt to model the transition dynamics of the environment, nor does it attempt to explicitly learn the value of different states or actions. Instead, it iteratively attempts to improve a parameterized policy. When function approximation is used, direct policy search is related to approximate policy iteration.

Direct policy search can be broken down into gradient-based methods, also known as policy gradient methods, and methods that do not rely on the gradient. In policy gradient methods, often the gradient is not explicitly known and must be approximated via sampling from the environment. An example of these methods is the REINFORCE algorithm [49]. Gradient-free methods include evolutionary algorithms. A survey on policy search and applications to robotics is [6].

2.5 Value-Based Methods

In value-based reinforcement learning methods, an agent learns the expected reward r conditioned on a particular action a in a certain state s. This defines a function which is called the *action-value function*.

A central theme in many reinforcement learning algorithms is *temporal-difference* (TD) learning. In value-based TD learning, an approximation of the action-value function is used as an initial estimate and is compared to sample returns obtained from the environment. Iterative updates are then made based on the observed error, called the *TD error*. This is an example of *bootstrapping*.

A distinction is made between *on-policy* and *off-policy* reinforcement learning algorithms. On-policy algorithms learn the value of the policy that is actually carried out by an agent, which includes exploration steps that it may take, whereas off-policy algorithms learn the value of an optimal policy that is independent of the actual policy being followed.

Q-learning [47] is a commonly used off-policy value-based reinforcement learning algorithm. In Q-learning, we define an action-value function $Q(s, a)$ and define an iterative update procedure called *one-step Q-learning* [44, Chapter 6] as follows:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \Big(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \Big)$$

where α is a learning-rate parameter, and γ is a parameter that *discounts* the value of future rewards versus present rewards.

3 Reinforcement Learning with Neural Networks

3.1 Introduction

As mentioned previously, neural networks can be used as non-linear function approximators. They are well suited to problems where the state space is high-dimensional, due to their frequent ability to effectively learn to detect and exploit patterns in complex domains. The use of neural networks to compress a high-dimensional state space into a compact representation that is used to learn a control policy by a reinforcement learning algorithm is also referred to as *deep reinforcement learning.*

3.2 Neural Fitted Q Iteration

In *batch reinforcement learning* [48, Chapter 2], a sequence of samples is collected from the environment in order to perform batch learning. Using a neural network function approximator with batch learning led to the *Neural Fitted Q-Iteration* (NFQ) algorithm, which is described in detail in [30] and is an extension of *experience replay* methods [24]. A practical guide to applying the algorithm is provided in [31].

3.3 Deep Q-Networks

While NFQ is a batch reinforcement learning method, it is also possible to construct an online algorithm for Q-learning using neural networks. Furthermore, a convolutional neural network for compressing high-dimensional input images can be integrated into the overall network that approximates the action-value function. These are the main extensions used in the *Deep Q-Network* (DQN) algorithm [27, 28]. Furthermore, this algorithm introduces the notion of a separate *target network* that increases the stability of online reinforcement learning with neural networks by reducing correlations between updates to the parameters and the targets used to calculate those updates.

3.4 Regularization

Regularization in machine learning consists of any method which aims to decrease the generalization error of a model without decreasing the training error [3, Chapter 5].

Regularization techniques have been studied extensively in machine learning in general, but have received less attention within the reinforcement learning literature. Recent work includes [8] and [25]. An excellent overview of the generalization ability of neural network models grounded in Solomonoff's algorithmic probability [41] and Kolmogorov complexity [18] is presented in [37]. Dropout [43] has been found to be a simple and effective regularization method for neural networks.

In [19], it is mentioned that combining regularization with NFQ can be problematic when there is a minimal amount of training data, as it can cause samples with low frequency to be regularized away despite carrying valuable information.

Regularization is also applied in *autoencoder* [3, Chapter 14] neural network models, in particular in *denoising autoencoders* and *sparse autoencoders*, which can be used to compress the input to reinforcement learning algorithms.

The original NFQ and DQN algorithms do not incorporate regularization into their neural network models.

4 Regularized Convolutional Neural Fitted Q Iteration

4.1 Introduction

We introduce an algorithm that combines ideas from the Neural Fitted Q Iteration and Deep Q-Network algorithms and adds dropout regularization, resulting in a novel variation which we call *Regularized Convolutional Neural Fitted Q Iteration* (RC-NFQ).

4.2 Algorithm

The *Regularized Convolutional Neural Fitted Q Iteration* (RC-NFQ) procedure is illustrated in Algorithm 1. The hyperparameters for the algorithm are described in Table 1. In addition to the hyperparameters, a suitable choice of architecture for the convolutional neural network that will serve both as a function approximator and to learn the action-value function must be selected. In the following section, we will describe an example of such an architecture.

5 Architecture

5.1 Q-network

We consider the setting in which an agent is embedded in an environment and is equipped with a two-dimensional visual sensor, such as a video camera or a

Algorithm 1 Estimate the action-value function using a convolutional neural network with dropout regularization

```
Input:
        $E$ is an environment, specifying the space of states, actions and rewards
        $C$ is an architecture for a convolutional neural network with dropout
            regularization layers
        $hyperparameters$, described in Table 1
Output:
        Returns a learned action-value function $Q$
 1: procedure RC-NFQ($E, C, hyperparameters$)
 2:     Parameterize identical convolutional neural network models $Q_0$ and $\hat{Q}_0$ which
            will be used to learn to approximate the action-value function, using the
            architecture specified in $C$ and the dropout probability $\alpha_{drop}$ for the dropout
            regularization layers
 3:     Initialize action-value function $Q_0$ with a parameter vector $\theta$ of random initial
            weights
 4:     Initialize target action-value function $\hat{Q}_0$ with $\theta^- = \theta$
 5:     Initialize experience replay buffer $D$
 6:     for episode $i = 0, \alpha_{eps}$ do
 7:         Initialize temporary experience buffer $\tilde{D}$
 8:         for $t = 1, \alpha_{len}$ do
 9:             Select random action with probability $\epsilon$
10:             Otherwise, select action $a$ maximizing the action-value function $Q_i(s, a)$
11:             Execute action $a$ in environment $E$ and observe $r$ and $s'$
12:             Store transition $(s, a, r, s')$ in $\tilde{D}$
13:         end for
14:         Append $\tilde{D}$ to the experience replay buffer $D$
15:         for iteration $k = 0, \alpha_{iters}$ do
16:             Sample random batch of $\alpha_{samples}$ from $D$ and store in $D'$
17:             Generate a pattern set of training targets where
                    $y_i = D'^r + \gamma\hat{Q}_i(D'^{s'}, D'^a)$
18:             Call RMSprop with learning rate $\alpha_{lr}$ to perform gradient descent on
                    $(y_i - Q_i(D'^s, D'^a; \theta))^2$ and store the updated parameters in $Q_{i+1}$
19:             if k is a multiple of $\alpha_{freq}$ then
20:                 Update target action-value function $\hat{Q}_{i+1}$ with parameters from $Q_i$
21:             else
22:                 Copy the parameters from $\hat{Q}_i$ to $\hat{Q}_{i+1}$
23:             end if
24:         end for
25:     end for
26:     return $Q_{\alpha_{eps}}$
27: end procedure
```

Table 1. Hyperparameters for the RC-NFQ algorithm

Hyperparameter	**Description**
γ	discount factor for future rewards
α_{lr}	learning rate for RMSprop
α_{freq}	frequency at which the target Q-network is updated
α_{iters}	number of iterations of fitted Q-iteration to run between episodes
α_{len}	length of each episode
α_{eps}	number of episodes
$\alpha_{samples}$	number of samples to use within each iteration of fitted Q-iteration
α_{drop}	dropout probability for the dropout regularization layers

series of screen captures from a simulated environment. A convolutional neural network is used to compress the input images.

We will now present an example of a suitable architecture for the Q-network for the RC-NFQ algorithm. The convolutional layers are similar to those used in [27] and [26] with several modifications. The state input consists of only one input image, and the action is also fed as input, encoded as a one-hot vector. We add dropout layers after the first and second convolutional layers, and after the first fully connected layer. The dropout probability used is $p = 0.25$. The details of the convolutional neural network are shown in Table 2, and the architecture of the Q-network is shown in Table 3.

The total number of parameters to be trained in this example architecture is 305713. The network is trained using RMSprop.

5.2 Source Code

We have developed an initial implementation of the RC-NFQ algorithm with the architecture described here using the *Keras* deep learning library [4]. It is available at: https://github.com/cosmoharrigan/rc-nfq.

Table 2. Example architecture for the convolutional neural network

Layer	**Input**	**Output**	**Filter Width**	**Filters**	**Stride**	**Parameters**
Convolutional 1	(1, 64, 64)	(16, 15, 15)	8	16	(4, 4)	1040
Activation 1	(16, 15, 15)	(16, 15, 15)	-	-	-	-
Dropout 1	(16, 15, 15)	(16, 15, 15)	-	-	-	-
Convolutional 2	(16, 15, 15)	(32, 6, 6)	4	32	(2, 2)	8224
Activation 2	(32, 6, 6)	(32, 6, 6)	-	-	-	-
Dropout 2	(32, 6, 6)	(32, 6, 6)	-	-	-	-
Flatten	(32, 6, 6)	(1152)	-	-	-	-

Table 3. Example architecture for the Q-network

Layer	**Input**	**Output**	**Parameters**
ConvNet	(1, 64, 64)	(1152)	9264
Action Input	(4)	(4)	-
Merge	(1152) and (4)	(1156)	-
Dense 1	(1156)	(256)	296192
Activation	(256)	(256)	-
Dropout	(256)	(256)	-
Dense 2	(256)	(1)	257

6 Possible Extensions

6.1 Algorithm Modifications

An immediate extension of the convolutional neural network architecture with dropout layers as described would include the addition of an LSTM [13] recurrent neural network layer to allow the controller to take advantage of memory of prior states. In order to extend the integration of dropout regularization layers with the NFQ algorithm and convolutional neural networks to LSTM networks, the improved methods for effectively combining dropout with RNNs presented in [10] could be applied.

An alternative simpler way to add a minimal amount of memory would be to provide the network with a sequence of several recent frames as an input tensor to the convolutional neural network, as in [26–28].

Extending the RC-NFQ algorithm to the online case, resulting in a variant of the Deep Q-Network (DQN) algorithm utilizing dropout regularization, would also be a possible technique. Modifying the Q-network further by adding batch normalization layers [16] and analyzing their effect on learning in both the NFQ and DQN settings could also be worthwhile.

In [36], more sophisticated techniques for sampling from an experience replay memory are analyzed, and could be tested in this setting as well.

Actor-critic [44, Chapter 6] methods and advantage learning [2, 12, 46, 39] would be another worthwhile direction in which to extend the RC-NFQ algorithm. Additionally, Monte-Carlo tree search methods have recently been combined with convolutional neural networks for control in *TORCS* [11] and *Go* [40] and present a promising research direction.

6.2 Exploration Strategies

Improved exploration strategies beyond epsilon-greedy may help improve the efficiency of learning.

A simple addition to the exploration strategy would be to implement softmax action selection rather than winner-take-all action selection based on the estimated Q-values.

More complex additions would include the addition of an intrinsic motivation function which rewards the agent for discovering novel states; in [5], the agent was rewarded for discovering states that it did not yet know how to compress well, with the intention of driving the compressor to improve its performance over time, and by extension, to improve the ability of the controller to find a more effective control policy.

In recent work [29], an approach to deep (temporally-extended) exploration was applied to DQN and could also be extended to the RC-NFQ setting.

7 Conclusion

A brief overview of neural networks, reinforcement learning and the deep reinforcement learning paradigm was presented. This was followed by a summary of two value-based methods, Neural Fitted Q Iteration (NFQ) and Deep Q-Networks (DQN).

A detailed algorithm was then described implementing a novel extension of NFQ called *Regularized Convolutional Neural Fitted Q Iteration* (RC-NFQ), which adds convolutional neural networks and dropout regularization to the NFQ algorithm along with several additional elements from the DQN algorithm.

Finally, several additional extensions were proposed to improve the learning algorithm and exploration strategies.

References

1. Anselmi, F., Poggio, T.: Representation learning in sensory cortex: a theory. Tech. rep., Center for Brains, Minds and Machines (CBMM) (2014)
2. Baird III, L.C.: Advantage updating. Tech. rep., DTIC Document (1993)
3. Bengio, Y., Goodfellow, I.J., Courville, A.: Deep Learning (2015)
4. Chollet, F.: Keras: Deep learning library for theano and tensorflow. https://github.com/fchollet/keras (2015)
5. Cuccu, G., Luciw, M., Schmidhuber, J., Gomez, F.: Intrinsically motivated neuroevolution for vision-based reinforcement learning. In: 2011 IEEE International Conference on Development and Learning (ICDL). vol. 2, pp. 1–7. IEEE (aug 2011)
6. Deisenroth, M.P., Neumann, G., Peters, J., et al.: A survey on policy search for robotics. Foundations and Trends in Robotics 2(1-2), 1–142 (2013)
7. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research 12, 2121–2159 (2011)
8. Farahmand, A.M., Ghavamzadeh, M., Mannor, S., Szepesvári, C.: Regularized policy iteration. In: Advances in Neural Information Processing Systems. pp. 441–448 (2009)
9. Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics 36(4), 193–202 (1980)

10. Gal, Y.: A theoretically grounded application of dropout in recurrent neural networks. arXiv preprint arXiv:1512.05287 (2015)
11. Guo, X., Singh, S., Lee, H., Lewis, R.L., Wang, X.: Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In: Advances in Neural Information Processing Systems. pp. 3338–3346 (2014)
12. Harmon, M.E., Baird III, L.C.: Multi-player residual advantage learning with general function approximation. Wright Laboratory, WL/AACF, Wright-Patterson Air Force Base, OH pp. 45433–7308 (1996)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
14. Hubel, D.H., Wiesel, T.N.: Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology 195(1), 215–243 (1968)
15. Hutter, M.: Universal Artificial Intelligence, vol. 1 (2005)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
17. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
18. Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Problems of information transmission 1(1), 1–7 (1965)
19. Lampe, T., Riedmiller, M.: Approximate model-assisted neural fitted q-iteration. In: Neural Networks (IJCNN), 2014 International Joint Conference on. pp. 2698–2704. IEEE (2014)
20. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation 1(4), 541–551 (dec 1989)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2323 (1998)
22. Legg, S.: Machine super intelligence. Ph.D. thesis, University of Lugano (2008)
23. Legg, S., Hutter, M.: Universal intelligence: A definition of machine intelligence. Minds and Machines 17(4), 391–444 (2007)
24. Lin, L.J.: Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine Learning 8(3-4), 293–321
25. Liu, B., Mahadevan, S., Liu, J.: Regularized off-policy td-learning. In: Advances in Neural Information Processing Systems. pp. 836–844 (2012)
26. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783 (2016)
27. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
28. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.a., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
29. Osband, I., Blundell, C., Pritzel, A., Van Roy, B.: Deep exploration via bootstrapped dqn. arXiv preprint arXiv:1602.04621 (2016)
30. Riedmiller, M.: Neural fitted q iteration first experiences with a data efficient neural reinforcement learning method, Lecture Notes in Computer Science, vol. 3720. Springer Berlin Heidelberg, Berlin, Heidelberg (oct 2005)

31. Riedmiller, M.: 10 Steps and Some Tricks to Set up Neural Reinforcement Controllers. In: Neural Networks: Tricks of the Trade, pp. 735–757. Springer (2012)
32. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: Neural Networks, 1993., IEEE International Conference on. pp. 586–591. IEEE (1993)
33. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., DTIC Document (1985)
34. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision 115(3), 211–252 (apr 2015)
35. Russell, S., Norvig, P., Intelligence, A.: A modern approach. Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25, 27 (1995)
36. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
37. Schmidhuber, J.: Discovering Neural Nets with Low Kolmogorov Complexity and High Generalization Capability. Neural Networks 10(5), 857–873 (jul 1997)
38. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks 61, 85–117 (2015)
39. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)
40. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature 529(7587), 484–489 (2016)
41. Solomonoff, R.J.: A formal theory of inductive inference. Part I. Information and control 7(1), 1–22 (1964)
42. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
43. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)
44. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
45. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning 4, 2 (2012)
46. Wang, Z., de Freitas, N., Lanctot, M.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
47. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine learning 8(3-4), 279–292 (1992)
48. Wiering, M., van Otterlo, M.: Reinforcement Learning: State-of-the-art, vol. 12. Springer Science & Business Media (2012)
49. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8(3-4), 229–256 (1992)
50. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)