Learning Deep State Representations With Convolutional Autoencoders

Gabriel Barth-Maron
Supervised by Stefanie Tellex

Department of Computer Science
Brown University

Abstract

Advances in artificial intelligence algorithms and techniques are quickly allowing us to create artificial agents that interact with the real world. However, these agents need to maintain a carefully constructed abstract representation of the world around them [9]. Recent research in deep reinforcement learning attempts to overcome this challenge. Mnih et al. [24] at DeepMind and Levine et al. [18] demonstrate successful methods of learning deep end-to-end policies from high-dimensional input. In addition, Böhmer et al. [1] and Mattner et al. [22] extract deep state representations that can be used with traditional value function approximation algorithms to learn policies. We present a model that discovers low-dimensional deep state representations in a similar fashion to the deep fitted Q algorithm [1]. A plethora of function approximation techniques can be used in the lower dimension space to obtain the Q-function. To test our algorithms, we run several experiments on 80×20 images taken from a 10×2 grid world and show that convolutional autoencoders can be trained to obtain deep state representations that are almost as good as knowing the ground-truth state.

1 Introduction

Reinforcement learning provides an excellent framework for planning and learning in non-stochastic domains. Since inception it has been used to accomplish a wide variety of tasks, from robotics [10, 5, 15] to sequential decision-making games [32, 11], and dialogue systems [27, 34].

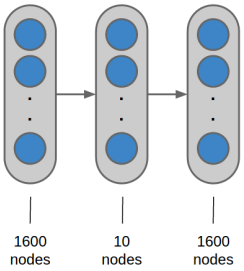
However, many reinforcement learning algorithms have a run-time that is polynomial in the number of states and actions. To learn in large domains, researchers have had to carefully craft features of their state space so that they are general enough to represent the original problem, but small enough to be computationally tractable.

Figure 1: A 80×20 gray scale image of the 10×2 state. The agent is at location $(3, 0)$ and the goal is at location $(9, 1)$.

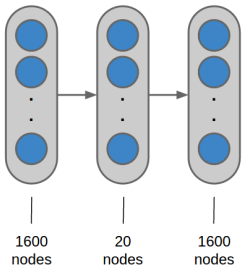
Feature engineering becomes a major hindrance as we create learning agents for more complex state spaces. Additionally, it requires expert knowledge and does not generalize well across different domains. Several areas of research attempt to deal with the challenge of exponentially large state spaces, such as Monte Carlo Tree Search [2], hierarchical planning [4, 30, 7], and value function approximation [29].

Here we take an alternative approach with a focus on planning with sensor input. Visual information is an easily accessible rich source of information, however uncovering structured information is a difficult and well studied problem in computer vision. Many vision problems have been solved through the use of carefully crafted features such as scale invariant feature transformations [20] and histogram of gradients [3]. Recent advances in deep learning have made it possible to automatically extract high-level features from raw visual data, leading to breakthroughs in several areas of computer vision [14, 26, 23].

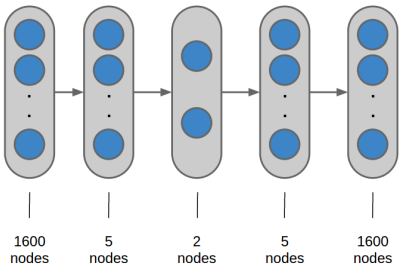
In our model we use neural networks as an unsupervised technique to learn an abstract feature representation of the raw visual input. Similar to hierarchical techniques, these neural networks allow us to plan in the (significantly simplified) abstract state space. This model is similar to the algorithm designed by DeepMind that plays Atari 2600 games from visual input [24]. However their algorithm performs end-to-end learning (which directly produces a policy), whereas ours learns a deep state representation that can be used by a variety of reinforcement learning algorithms. In addition, the DeepMind algorithm does not allow for model-based alternatives, as we believe ours does. Böhmer et al. [1], Mattner et al. [22] have created a deep fitted Q (DFQ) algorithm that is very similar to what we propose, however our use



(a) Autoencoder AE-10 with 10 hidden nodes.



(b) Autoencoder AE-20 with 20 hidden nodes.



(c) Stacked autoencoder SAE with 2 final hidden nodes.

Figure 2: Autoencoder architectures

of convolutional autoencoders takes advantage of image structure and produces better state representations.

We used 80×20 pixel gray scale images taken from a 10×2 grid world, an example state may be seen in Figure 1. Because the 10×2 grid world can be characterized by only two numbers – the agent's x and y coordinates – one of our goals is to attempt to compress these images to a two dimensional output.

In Section 2 we give a brief overview of reinforcement learning and deep learning. Section 6 reviews state of the art techniques that combine reinforcement learning and deep learning. Then in Section 3 we introduce our models, and show their empirical performance in Sections 4 and 5.

2 Background

This section should serve as a self-contained introduction to reinforcement learning and deep learning for those who are not already familiar with the fields.

2.1 Reinforcement Learning

Reinforcement learning problems are typically modelled as a Markov Decision Process (MDP). A MDP is a five-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a state space; $\mathcal{A}$ is the agent's set of actions; $\mathcal{T}$ denotes $\mathcal{T}(s' \mid s, a)$, the transition probability of an agent applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and arriving in $s' \in \mathcal{S}$; $\mathcal{R}(s, a, s')$ denotes the reward received by the agent for applying action a in state s and transitioning to state s'; and $\gamma \in [0, 1]$ is a discount factor that defines how much the agent prefers immediate rewards over future rewards (the agent prefers to maximize immediate rewards as γ decreases). MDPs may also include terminal states that cause all action to cease once reached.

Reinforcement learning involves estimating a value function from experience, simulation, or search [28, 33]. Typically the value function is parametrized by the state space – there exists one unique entry per state. However in continuous state spaces (or as we will later see, in large discrete state spaces) it is desirable to find an alternate parametrization of the value function. The most common technique for doing so is linear value function approximation, where the value function is represented as a weighted linear sum of a set of features [12]. These features are also known as basis functions, some common examples being Radial Basis Functions, CMACs, and the Fourier Basis Function.

One particular algorithm for learning a linear value function approximation is Gradient Descent SARSA(λ) [25]. This algorithm combines Q-learning with Temporal Difference learning (TD-learning) to learn the Q-function [1]. Lin [19] derives an update equation for a Q-learning algorithm that uses a neural network basis function (it is also applicable to any other basis function) with weights w.

$$\Delta w_t = \eta \left[r_t + \gamma \max_{a \in \mathcal{A}} Q_{t+1}(a) - Q_t \right] \frac{\partial Q_t)}{\partial w_t} \qquad (1)$$

The Gradient Descent SARSA(λ) update scheme is similar with two notable exceptions. First, in order to update previous states Δw_t is multiplied by a weighted sum of previous gradients. Second, the max operator is dropped in favor of using Q_{t+1} associated with the action that was selected, which allows for a better trade-off between exploration and exploitation – as the algorithm converges it will start behaving as if it were always selecting the action that maximizes the Q-function.

2.2 Deep Learning

An autoencoder is a fully-connected neural network that attempts to learn the identity function. Additionally the network contains a single hidden layer that has a number of nodes significantly less than the input. During training the autoencoder attempts to find a good compression of the input data. In addition, autoencoders can be stacked – the output of one autoencoder's hidden layer as the input of another – to form deep architectures. Autoencoders and stacked autoencoders have been shown to be very useful in performing unsupervised dimensionality reduction [8].

Convolutional neural networks (CNNs) use convolution to take advantage of the locality of image features. In addition, since these networks share the kernel's weights for each layer, they are be much sparser than their fully-connected counterparts. CNNs have

[1] We use Q_t as shorthand for $Q(s_t, a_t)$.

been used to achieve state of the art performance in image classification [14], face verification [31], and object detection [17].

3 Architectures

We used autoencoders to learn abstract features for images similar to the one in Figure 1 in an unsupervised manner. To train these networks we used backpropagation on an image set that captures the entirety of the state space. We combined different numbers of layers and hidden nodes, and have reported the results for some of the final models in Section 5. We also used convolutional autoencoders (CAEs) to take advantage of the structure and locality that is found in naturally occurring images.

The output of the middle layer of the (convolutional) autoencoders was used as a basis function, which served as the features for linear value function approximation. Value function approximation then greedily constructed a policy that the agent can follow.

3.1 Autoencoders

Autoencoders were an obvious choice because we wanted to learn abstract features from images in an unsupervised manner. Backpropagation with a euclidean loss function was used to train the autoencoders in Figure 2. In addition, to train the stacked autoencoder in Figure 2(c) we used layer-wise pre-training to obtain good initialization for its hidden layers. These weights were then fine-tuned all together. Both the pre-training and fine-tuning steps used the entire state-image data set.

Figures 2(a) and 2(b) show two autoencoder architectures with a single layer containing 10 and 20 hidden nodes respectively. Figure 2(c) is a stacked autoencoder whose middle-most hidden layer has only two nodes.

3.2 Convolutional Autoencoders

The CAEs use a similar architecture to those described in [21], where the weights for the deconvolutional layers are "tied" and are the transpose of the weights for the convolutional layers. The main difference with the neural networks we created is that no pooling or unpooling layers were used. Pooling layers discard useful location information [18], which would have prevented our system from learning features for detecting the agent's location.

The CAEs in Figures 3(a) and 3(b) both used two 8×10 kernels (one for each feature map) with a 8×10 stride. It is important to point out that this neural network is over-engineered to the 80×20 image problem, and one of the kernels quickly became an accurate "agent" detector.

On the other hand, the CAEs in figures 3(c) and 3(d) both used more general kernels. The first layer used a 5×5 kernel with a 3×3 stride, while each of the subsequent convolutional layers each used a 3×3 kernel. The difference between SCAE-8 and SCAE-4 was the number of features maps in each. As we will discuss later, while more feature maps gave the SCAE-8 more accurate image compression, the increased feature set created a more difficult optimization problem for the value function approximation algorithm.

4 Experiments

All of our experiments used 80×20 images taken from a 10×2 grid world as seen in Figure 1. The autoencoders AE-10, AE-20, SAE, along with the convolutional autoencoders CAE and SCAE-AGENT were trained on all 20 possible images while the goal was at location $(9, 1)$. The convolutional autoencoders SCAE-8 and SCAE-4 were both trained on all 400 possible images by moving both the agent and goal. The larger training data set was used to make the kernels goal-location invariant.

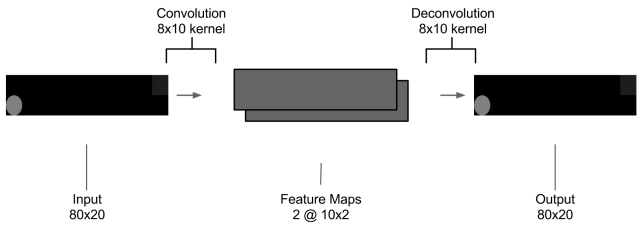
The middle layer of each of these neural networks was then used as a feature basis for Gradient Descent SARSA(λ) with $\gamma = 0.99$, $\lambda = 0.5$, and the value function was initialized to 1.0. The reward function was set to -1 everywhere and the agent would terminate upon finding the goal. Finally, the learning rate was different for each model and is listed in Table 1. The learning rates were obtained by setting them as high as possible while ensuring Gradient Descent SARSA(λ) converged on a policy.

We ran the algorithm for a total of 1,000 episodes. An episode began with the agent in a starting position and ended when either when the agent reached the goal or 10,000 steps had been taken. As our baseline we used a Fourier Basis function [12], which used the agent's exact x and y location. Because the baseline used the agent's known location it gave us a very high upper bound for performance. In addition, while we chose to use Gradient Descent SARSA(λ), any value function approximation algorithm could have been used in its place.

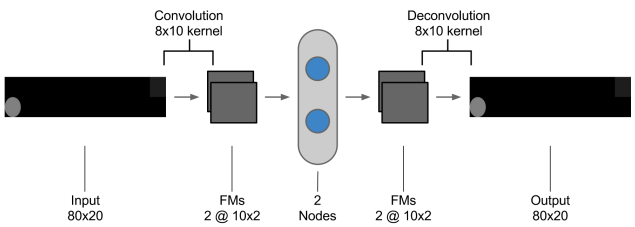
Model	Learning Rate
AE-10	0.002
AE-20	0.002
SAE	0.005
CAE	0.2
SCAE-AGENT	0.0006
SCAE-16	0.002
SCAE-32	0.005

Table 1: Learning rates for the different models.

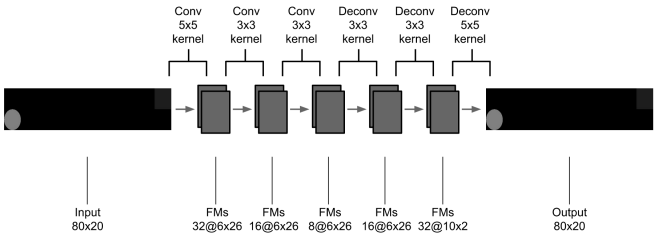
Each trial consisted of 1,000 episodes and was run 10 times. We reported the average quantities along with their 95% confidence interval in Figure 4. In Figures 4(a) and 4(c) the number of steps the agent took in each learning episode is plotted. As expected this number significantly drops off after the first few iterations for all models except CAE. Figures 4(b) and 4(d) show the total reward accumulated across all learning episodes. Note that the derivative of this chart is the reward earned at each episode.



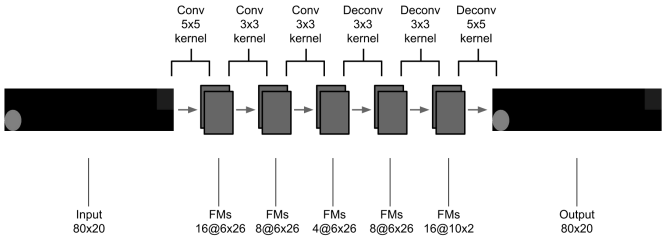
(a) Convolutional Autoencoder CAE.



(b) Stacked Convolutional Autoencoder SCAE-AGENT.



(c) Stacked Convolutional Autoencoder SCAE-8.



(d) Stacked Convolutional Autoencoder SCAE-4.

Figure 3: Convolutional Autoencoder architectures

5 Results

As we see from Figures 4(a) and 4(b) AE-10 and AE-20 both converged quickly upon the same optimal policy that the Fourier Basis model learned. However, the stacked autoencoder SAE was unable to learn the optimal policy and learned some other sub-optimal one instead. Its sub-optimal performance is likely due to the inability of the stacked autoencoder to learn a good deep state representation for our 10×2 grid world. The fully-connected nature of the stacked autoencoder made the optimization problem too difficult for backpropagation to train over our data set.

In Figures 4(c) and 4(d) we have compared all of the convolutional autoencder models and the two best autoencoders AE-10 and AE-20. The CAE model outperformed all of them and converged to the optimal policy almost as quickly as the Fourier Basis. This was an impressive accomplishment given that the Fourier Basis model had direct access to the agent's x and y locations; indicating that the CAE network learned a very good deep state representation. All the other convolutional models outperformed the autoencoders AE-10 and AE-20, which is expected given how well convolution has performed on computer vision related tasks.

The SCAE-AGENT and SCAE-8 models performed worse than the CAE and CAE-4 networks, however for different reasons. Because the SCAE-AGENT model had a hidden layer with only 2 nodes it was harder to train and did not achieve as low a training error as the other convolutional models. This difficulty resulted in a sub-par deep state representation that Gradient Descent SARSA(λ) was unable to learn over. On the other hand SCAE-8 achieved very low training loss, however its deep state representation was large enough (its middle layer contains 352 neurons) that it made the optimization problem much more difficult for Gradient Descent SARSA(λ).

We also analyzed the performance of our models during the first 100 episodes. In Figure 5 we have charted the number of steps the different models took during this time period. It is interesting to notice how quickly all of these models converged. Almost all of them had the same performance as the Fourier Basis model after only 20 episodes, and after 70 episodes they all seem to be identical. What varies did vary is the number of steps taken before they converge, which helps to explain the variation we see in Figure 4(d).

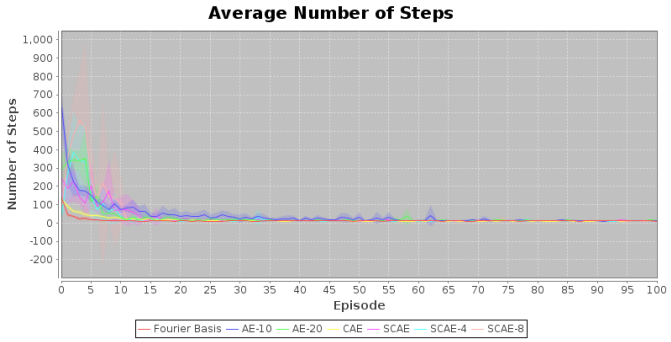
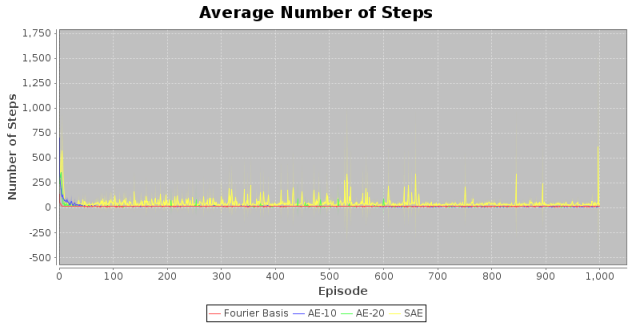
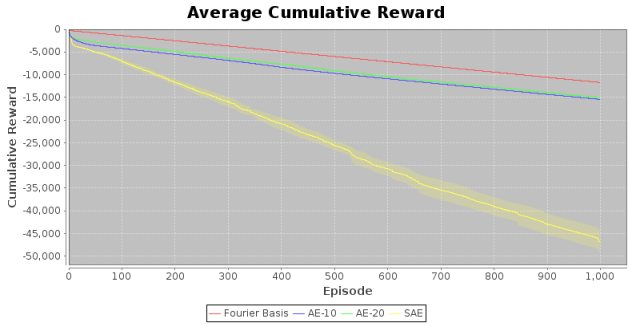


Figure 5: A closer look at the number of steps taken in the first 100 episodes for the convolutional autoencoders.

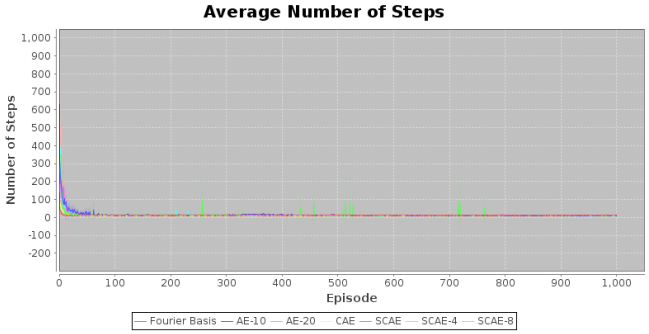
Finally, we wanted to see how our models behaved to changes in the goal location. As we mentioned earlier, the SCAE-4 and SCAE-8 neural networks had been trained on the full 400 image data set obtained by moving both the agent and goal, while the other models had only been trained on the 20 images obtained by moving only the agent. To our surprise, changing the goal's location (and the reward function) did not significantly affect the performance of any of the models. This goal-location



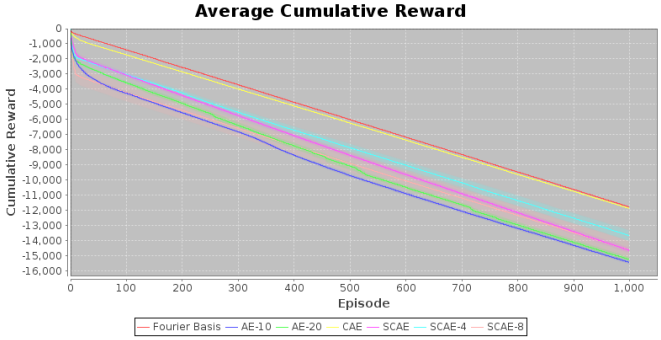
(a) The average number of steps taken per episode for different autoencoders.



(b) The cumulative reward received over all episodes for different autoencoders.



(c) The average number of steps taken per episode for different convolutional autoencoders.



(d) The cumulative reward received over all episodes for different convolutional autoencoders.

Figure 4: Results from using Gradient Descent SARSA(λ) with the hidden features of several different neural network architectures.

invariance is an interesting result and shows that the AE-10, AE-20, CAE, and SCAE models learned features that were able to accurately represent the agent's location despite changes in the goal-location.

6 Related Work

With the recent resurgence of deep learning there has been significant work in combining deep learning and reinforcement learning. This work can be separated into two categories: end-to-end learning and deep state representation [1]. End-to-end learning takes a direct approach and learns a non-linear policy (value function) directly from the input data. On the other hand, deep state representation first finds a reduced dimension representation for the input data, and then uses an approximation technique to obtain an approximate Q-function.

While end-to-end learning results in direct policies (without the need to find a reduced dimension state representation) it requires a large amount of data [1]. This requisite makes it difficult or even impossible to train on robots or other environments where taking a large number of samples is not possible.

6.1 End-to-End Learning

In Mnih et al. [24] the raw video feed from Atari 2600 games is fed into a convolutional network that outputs the Q-value for each possible action. The loss function for the aptly named Deep Q-Network is a function of the expected Q-value for the input state. This architecture tightly integrates deep learning and reinforcement learning and achieved super human performance on many of the games.

The authors of Levine et al. [18] also use a convolutional network, which receives a video feed from a PR2 robot and creates a controller that can be used to pick up objects. Their network outputs a probability distribution over actions, which can be used as a controller for a robotic arm. There are several training phases, first the convolutional layers of the network are pre-trained on visual data. Next there is a supervised training phase, in which the robot knows the location of the object. Finally the robot attempts to use the network to pick up and manipulate objects, the result of its actions are used as reinforcement to update the network in an unsupervised manner.

6.2 Deep State Representation

The work done by Mattner et al. [22] to balance an inverted pendulum, and Lange et al. [16] to control a slot-car racer both use the Deep Fitted Q (DFQ) algorithm. The algorithm trains an autoencoder on the frames of a video feed that captures the entire scene. After training, the output of the middle hidden layer was used to approximate the Q-function. Any of a plethora of function

approximation algorithms may be used to perform this step. The DFQ algorithm is very similar to the models presented in this paper. However our use of convolutional neural networks allows us to take advantage of image locality and obtain better deep state representations.

6.3 Recurent Neural Networks

Finally there has been some work done on using Recurrent Neural Networks (RNNs) to combine deep learning and reinforcement learning. One notable example is Koutník et al. [13] who used evolutionary algorithms to evolve a network that was able to play the TORCS racing game. Similar to the previous papers, the RNN received frames from the video game and directly output the action that should be taken.

7 Conclusions

As reinforcement learning is called upon to tackle problems that rely on larger state spaces, deep learning will provide a useful way to perform dimensionality reduction and extract meaningful deep state representations. Throughout this paper we have examined the current algorithms and techniques that are being used to combine deep learning and reinforcement learning. We have presented an algorithm that, similar to the DFQ algorithm presented in Section 6.2, extracted deep state representations. One of our main contributions has been the use of CAEs to extract deep state representations. Finally, we tested several different architectures with Gradient Descent SARSA(λ) and reported their results.

There are several directions that this work can be taken. While almost any function approximation algorithm can be used to approximate the Q-function, it would be interesting to see how alternatives to Gradient Descent SARSA(λ) perform. In addition it would be very useful to compare the models presented here directly to those in Mattner et al. [22] and Lange et al. [16].

In Hafner and Riedmiller [6] an actor-critic algorithm is presented, in which both the policy and Q-function are represented by neural networks. Similarly, these deep state representations could be used in a model-based algorithm, where the model is also learned through deep learning.

Despite the fact that neural networks have been used in reinforcement learning for decades, combining reinforcement learning with deep learning is still a nascent field of research. It is not clear if end-to-end training or deep state representation is preferable, most likely each will have their own uses – analogous to the differences between value and policy iteration methods.

References

[1] Wendelin Böhmer, Jost Tobias Springenberg, Joschka Boedecker, Martin Riedmiller, and Klaus Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, pages 1–10, 2015.

[2] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000.

[5] Peggy Fidelman and Peter Stone. Learning ball acquisition on a physical robot. In *2004 International Symposium on Robotics and Automation (ISRA)*, page 6, 2004.

[6] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.

[7] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998.

[8] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[9] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721, 2013.

[10] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2619–2624. IEEE, 2004.

[11] Wolfgang Konen and Thomas Bartz-Beielstein. Reinforcement learning for games: failures and successes. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2641–2648. ACM, 2009.

[12] George Konidaris. Value function approximation in reinforcement learning using the fourier basis. 2008.

[13] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] Cody Kwok and Dieter Fox. Reinforcement learning for sensing strategies. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 3158–3163. IEEE, 2004.

[16] Sascha Lange, Martin Riedmiller, and A Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[17] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[19] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[20] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[21] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59. Springer, 2011.

[22] Jan Mattner, Sascha Lange, and Martin Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *Neural Information Processing*, pages 126–133. Springer, 2012.

[23] Volodymyr Mnih. *Machine learning for aerial image labeling*. PhD thesis, University of Toronto, 2013.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] Gavin A Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. 1994.

[26] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3626–3633. IEEE, 2013.

[27] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, pages 105–133, 2002.

[28] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1): 9–44, 1988.

[29] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063. Citeseer, 1999.

[30] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[31] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

[32] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3): 58–68, 1995.

[33] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[34] Jason D Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2): 393–422, 2007.