Neural Value Function Approximation in Continuous State Reinforcement Learning Problems

Martin Gottwald MARTIN.GOTTWALD@TUM.DE
Mingpan Guo GUO@FORTISS.ORG
Hao Shen SHEN@FORTISS.ORG

Abstract

Recent development of Deep Reinforcement Learning (DRL) has demonstrated superior performance of neural networks in solving challenging problems with large or continuous state spaces. In this work, we focus on the problem of minimising the expected one step Temporal Difference (TD) error with neural function approximator for a continuous state space, from a smooth optimisation perspective. An approximate Newton's algorithm is proposed. Effectiveness of the algorithm is demonstrated on both finite and continuous state space benchmarks. We show that, in order to benefit from the second order approximate Newton's algorithm, gradient of the TD *target* needs to be considered for training.

Keywords: TD Learning, Non-convex Optimisation, Approximate Newton's Algorithm

1. Introduction

Neural Network based Value Function Approximation (NN-VFA) is an important and effective instrument in Reinforcement Learning (RL) to handle large or infinite state spaces (Sutton and Barto, 1998; Bertsekas, 2012), demonstrated by successes in solving challenging problems in pattern recognition, computer vision, speech recognition, and game playing (LeCun et al., 2015; Yu and Deng, 2015; Mnih et al., 2015; Silver et al., 2017). Despite these advances, development of efficient NN-VFA based algorithms is still of great demand.

Recent attempts towards developing efficient NN-VFA methods follow the approach of developing gradient-based temporal difference algorithms (Sutton et al., 2009). The work in (Maei et al., 2009) adapts the development of the Gradient Temporal Difference (GTD) algorithms to a non-linear smooth VFA manifold setting. The proposed approach requires projections onto the VFA manifold, which is practically infeasible because the geometry of the VFA manifold is generally unavailable. A similarly approach developed in (Silver, 2013) projects estimates of the value function directly onto the vector subspace spanned by the weight matrices of the NNs. Due to its intrinsic complexity, no further analysis or numerical development has been delivered besides the original work. Such a difficulty in studying and developing NN-VFA methods is partially due to incomplete theoretical understanding of the optimisation of NNs. In this work, we propose to study this problem from the global analysis perspective, specifically in the framework of geometric optimisation (Absil et al., 2008), which naturally leads to an approximate Newton's (AN) algorithm.

Aside from some early work (Baird III, 1995), omitting gradients of the TD target has been a common practice for neural TD learning, e.g. (Riedmiller, 2005). The reasons being discussed include inferior learning speed (Baird III, 1995), limitation with non-Markovian feature space (Sutton et al., 2008), and non-differentiable operators, e.g. the max-operator in Q-learning. However, gradients of the TD target provide critical information about

the optimisation problem for developing efficient numerical algorithms. In this work, we investigate the first and second order derivatives of the TD error function evaluated at the global minima, and propose an efficient and effective Approximate Newton's algorithm.

2. Value Function Approximation with Feedforward Neural Networks

We model the RL process as a Markov Decision Process, which is a tuple $\mathcal{E}(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ consisting of a finite dimensional Euclidean state space $\mathcal{S}$, a countable set of actions $\mathcal{A}$, the unknown transition probabilities $s' \sim p(s, a)$, a scalar reward function $r(s)$ and a discount factor $\gamma \in (0, 1)$. We denote by $K \coloneqq \dim S$ the dimension of the state space.

The goal of the agent is to learn a *policy* π which maximises the expected reward. The policy can either be *deterministic* $\pi \colon \mathcal{S} \to \mathcal{A}$ or *stochastic* $\pi \colon \mathcal{S} \times \mathcal{A} \to [0, 1]$. In the latter case it is used to sample an action. The expected reward starting in state s and following the policy π afterwards is called *value function* and is defined as

$$V_\pi(s) \coloneqq \mathop{\mathbb{E}}_{s_1, s_2, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, s_{t+1}) \middle| s_0 = s \right] = \mathop{\mathbb{E}}_{s'} \left[r(s, s') + \gamma \cdot V_\pi\left(s'\right) \right], \tag{1}$$

where s' is the successor state of s after taking an action according to the policy π. Here, the second equality is referred to as the *Bellman's equation*. By treating V_π as a variable and using the right hand side of Eq. (1) as the *Bellman operator* T_π, the value function $V_\pi(s)$ is given as the unique solution of

$$V_\pi(s) = (\mathrm{T}_\pi V_\pi)(s), \quad \forall s \in \mathcal{S}. \tag{2}$$

For further details please refer to chapter one of (Bertsekas, 2012). The difference of both sides in Eq. (2) is the *one-step TD-error* $\delta(s, s') \coloneqq V_\pi(s) - (\mathrm{T}_\pi V_\pi)(s)$.

In high dimensional and continuous state spaces, an exact representation of the value function based on the discretisation of $\mathcal{S}$ is impossible. Therefore, efficient training of non-linear parameterised function approximation architectures is essential for estimating the actual value function of a policy. In this work, we obtain the approximator as the solution to a non-linear least squares problem. To ensure that V_π is the desired solution, we minimise for all $s \in \mathcal{S}$ the sum of squared TD-errors. The states s are obtained from roll-outs of the policy π. Since we allow for stochastic policies, we construct the performance objective in the following way

$$\mathcal{J} = \frac{1}{2} \mathop{\mathbb{E}}_{\substack{s \sim \mathcal{E}, a \sim \pi(s) \\ s' \sim p(s,a)}} \left[\delta(s, s')^2 \right] \approx \frac{1}{2T} \sum_{s, a, s' \sim \mathcal{E}} \Big(V_\pi(s) - r(s, s') - \gamma V_\pi(s') \Big)^2, \tag{3}$$

where $\mathcal{E}$ denotes the environment and T is the length of the roll-out.

In this work, we deploy the classic Feedforward Neural Networks (FNN) to approximate the value function V_π. The networks consist of L layers with n_l units per layer ($l = 1, \ldots, L$). The input layer has $n_0 = K$ units, the output layer only one, $n_L = 1$. As non-linearity $\sigma \colon \mathbb{R} \to \mathbb{R}$ an *unbounded*, *smooth* and *monotonically increasing* function (e.g. SoftPlus, Bent identity, etc.) is used. The last layer of the network employs the identity for σ. The *l-th layer mapping* is defined by stacking all units as $f_l(W_l, \phi_{l-1}) \coloneqq \sigma\left(W_l^\top \phi_{l-1}\right)$, with $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ being the l-th weight matrix. The bias vector b is an additional row of W_l, and the layer input ϕ_{l-1} is extended with a constant. The output can be obtained by

a recursive application of the layer mappings $\phi_l \coloneqq f_l(W_l, \phi_{l-1})$, starting with the input $\phi_0 \in \mathbb{R}^K$. Let $\boldsymbol{\mathcal{W}} \coloneqq \mathbb{R}^{K \times n_1} \times \ldots \times \mathbb{R}^{n_{L-1} \times 1}$ be the set of all weight matrices. Then the overall *Neural Value Function Approximation* is given as

$$f(\mathbf{W}, s) \coloneqq f_L(W_L, \cdot) \circ \ldots \circ f_1(W_1, s). \tag{4}$$

With such a construction, we can define the set of parameterised value functions as $\boldsymbol{\mathcal{F}} \coloneqq \{f(\mathbf{W}, \cdot) \colon \mathbb{R}^K \to \mathbb{R} | \mathbf{W} \in \boldsymbol{\mathcal{W}}\}$. With $\boldsymbol{\mathcal{F}}(K, n_1, \ldots, n_{L-1}, 1)$, we denote a concrete architecture of a FNN. By replacing V_π in Eq. (3) with $f(\mathbf{W}, \phi_0)$, we obtain the performance objective $\mathcal{J}(\mathbf{W})$ for training.

Now, we derive an AN algorithm to minimise $\mathcal{J}(\mathbf{W})$. First, we compute the directional derivative of $\frac{1}{2}\delta_{s,s'}^2$ at W_l in direction $H_l \in \mathbb{R}^{n_{l-1} \times n_l}$ for a single layer l

$$\begin{aligned}
\mathrm{D}_{W_l}\left(\frac{1}{2}\delta(s,s')^2\right)[H_l] =& \delta(s,s')\left(\mathrm{D}_{W_l} f(\mathbf{W}, s)\,[H_l] - \gamma \cdot \mathrm{D}_{W_l} f(\mathbf{W}, s')\,[H_l]\right) \\
=& \delta(s,s')\left(\psi_l^\top (I_{n_l} \otimes \phi_{l-1}^\top) - \gamma \cdot \psi_l'^\top (I_{n_l} \otimes \phi_{l-1}'^\top)\right) \cdot \mathrm{vec}(H_l),
\end{aligned} \tag{5}$$

where I_{n_l} is the identity matrix. Here, $\mathrm{D}_{W_l} f(\mathbf{W}, s)(H_l) = \Sigma_L W_L^\top \cdots W_{l+1}^\top \Sigma_l H_l^\top \phi_{l-1} =: \psi_l^\top H_l^\top \phi_{l-1}$, where $\Sigma_l \in \mathbb{R}^{n_l \times n_l}$ is a diagonal matrix consisting of the derivatives of σ w.r.t. its input. The prime indicates the usage of s' for ϕ_0. The operator $\mathrm{vec}(\cdot)$ stacks the columns of a matrix. The gradient of $\frac{1}{2}\delta_{s,s'}^2$ w.r.t. all weights $\mathbf{W}$ is then calculated as $\nabla_{\mathbf{W}} \frac{1}{2}\delta(s,s')^2 = \left[\nabla_{W_1} \frac{1}{2}\delta^2 \ldots \nabla_{W_L} \frac{1}{2}\delta^2\right]^\top$. We refer to (Shen, 2018) for more detailed derivations.

Then, we compute the second order directional derivative component-wise to obtain the Hessian, i.e., given two directions H_l and H_k for the layers l and k, we have

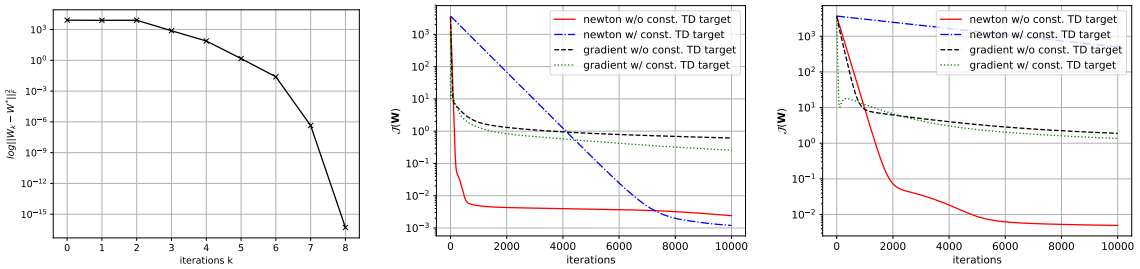
$$\begin{aligned}
\mathrm{D}_{W_k}\left(\mathrm{D}_{W_l}\frac{1}{2}\delta(s,s')^2[H_l]\right)[H_k] =& \mathrm{vec}(H_k)^\top \left(\psi_k^\top (I_{n_k} \otimes \phi_{k-1}^\top) - \gamma \psi_k'^\top (I_{n_k} \otimes \phi_{k-1}'^\top)\right)^\top \\
& \left(\psi_l^\top (I_{n_l} \otimes \phi_{l-1}^\top) - \gamma \psi_l'^\top (I_{n_l} \otimes \phi_{l-1}'^\top)\right) \mathrm{vec}(H_l) + \\
& \delta(s,s')\,\mathrm{D}_{W_k}\left(\mathrm{D}_{W_l} f(\mathbf{W}, s)[H_l] - \gamma\,\mathrm{D}_{W_l} f(\mathbf{W}, s')[H_l]\right)[H_k] \\
=& \mathrm{vec}(H_k)^\top J_k^\top J_l\,\mathrm{vec}(H_l).
\end{aligned} \tag{6}$$

At the global optimum $\mathbf{W}^*$, the TD-error vanishes and thus the summand with the second order derivatives can be neglected. We apply this approximation and avoid significant computational efforts. Automatic differentiation frameworks are not able to introduce such approximations on their own, thus a manual computation of the differential maps is necessary to reveal the special structure of the expression.

Next, due to the total differential, the overall second order directional derivative of $\mathcal{J}$ at $\mathbf{W}$ in direction $\mathbf{H} \in \boldsymbol{\mathcal{W}}$ can be computed as

$$\begin{aligned}
\mathrm{D}^2\mathcal{J}(\mathbf{W})[\mathbf{H}, \mathbf{H}] =& \sum_{l,k} \mathrm{vec}(H_k)^\top J_k^\top J_l\,\mathrm{vec}(H_l) \\
=& \left[\mathrm{vec}(H_1)^\top \ldots \mathrm{vec}(H_L)^\top\right] J^\top J \left[\mathrm{vec}(H_1) \ldots \mathrm{vec}(H_L)\right]^\top,
\end{aligned} \tag{7}$$

with $J = \mathrm{diag}\,(J_1 \ldots J_L)$. The Hessian of $\frac{1}{2}\delta_{s,s'}^2$ w.r.t. all weights $\mathbf{W}$ is finally given by $\mathsf{H}_{\mathbf{W}} \frac{1}{2}\delta(s,s')^2 = J^\top J$. As a last step, the gradients and Hessians for all (s,s')-tuples must



(a) *Seven-state Baird star* ($\alpha = 1$). (b) *Mountain car* ($\alpha = 0.01$). (c) *Mountain car* ($\alpha = 0.001$).

Figure 1: Convergence properties on finite and continuous benchmarks.

be averaged. The approximate Newton's descent direction $\eta \in \boldsymbol{\mathcal{W}}$ is now the solution of the regularised linear system

$$(\mathsf{H}_{\mathbf{W}}\mathcal{J} + c \cdot I) \cdot \eta = \nabla_{\mathbf{W}}\mathcal{J}, \tag{8}$$

where $c = 10^{-5}$ controls the strength. The regularisation is required since the approximation of the Hessian is only valid in a restricted neighborhood around the optimum.

3. Numerical Experiments

In this section, we provide empirical results of our approach, i.e., using the approximated Newton's algorithm to minimise the TD error without ignoring the TD target, from four different perspectives: 1) **convergence**, by evaluating the value function on a fixed batch of samples, 2) **generalisation**, by evaluating the influence of different neural network architectures for generalisation on the continuous state space, 3) **off-policy**, by evaluating the value of a policy from a different behavior policy, and 4) **policy iteration**, with a Q-learning style of policy improvement strategy, to investigate the performance on a full policy iteration setting. For experiments on convergence, off-policy, and policy iteration, we compare the cases of whether the gradient of the TD target is considered during training, with both first and second order approaches. As experiment cases, we use Baird star, mountain car, and the cartpole problem. The Baird star problem is a classic problem with finite state space that enables us to investigate the convergence property of the algorithm. Mountain car and cartpole are two classic reinforcement learning test cases with a typical continuous state space, and a manageable complexity that allows for an extensive investigation. In mountain car, the agent receives -1 reward every step if not in the goal area, and 0 if the goal area is reached. Whereas in cartpole, the agent receives 0 reward except when the pole falls, in which case a -1 reward is given.

Convergence We first demonstrate theoretical convergence properties using the seven-state Baird star problem. Each state is assigned with a two-dimensional random feature. We deploy an MLP architecture $\boldsymbol{\mathcal{F}}(2, 7, 1)$ with step size $\alpha = 1$. With a collection of $10,000$ interactions, we run the Approximate Newton's algorithm from randomly initialised weights. The convergence is measured by the distance of the accumulation point $\mathbf{W}^*$ to all iterates $\mathbf{W}^{(k)}$, i.e., by an extension of the Frobenius norm of matrices to collections of matrices as $\|\mathbf{W}^{(k)} - \mathbf{W}^*\|_F^2 := \sum_{l=1}^{L} \|W_l^{(k)} - W_l^*\|_F^2$. It is clear from Figure 1(a) that the

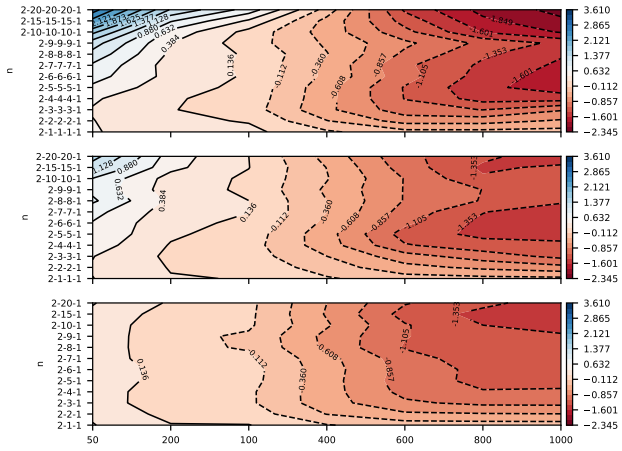


Figure 2: NN-VFA of different sample sizes (horizontal axis), architectures (bottom to up: wider and deeper). Red: lower error.

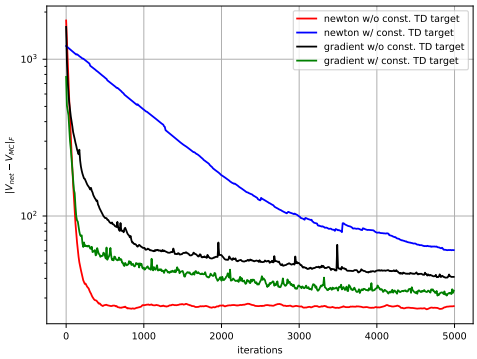


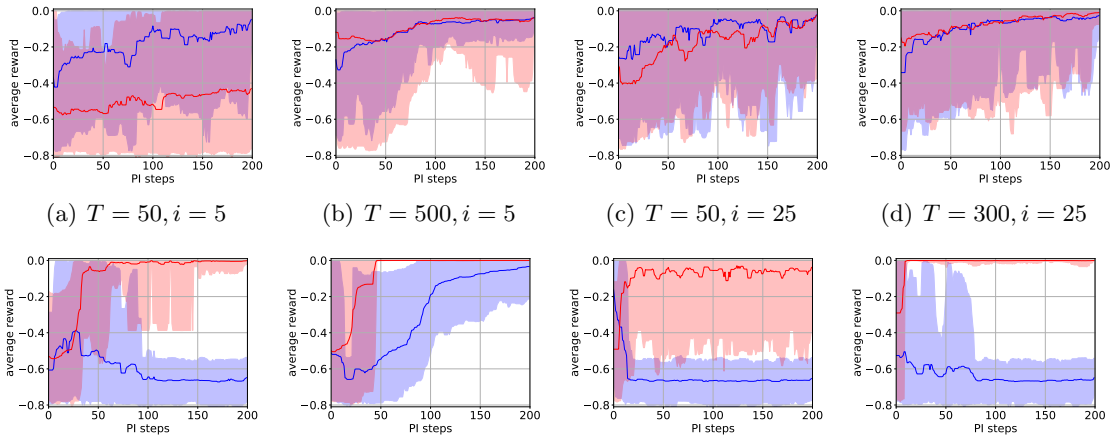
Figure 3: Off-policy policy evaluation for mountain car.

algorithm converges locally quadratically fast to a global minimiser, i.e. the ground truth value function.

We also test on the classic mountain car problem with a MLP architecture $\boldsymbol{\mathcal{F}}(2, 10, 10, 1)$, and observe convergent behavior as shown in Figure 1(b) and 1(c). It is clear from these Figures that considering the TD target for learning can benefit from the Approximate Newton's algorithm the most, and achieve a better final convergence [1].

Generalisation Continuous state space poses challenges for generalisation of neural value functions, as it can only be trained to fit a finite set of samples. It is widely believed that the architecture of NNs has an important influence on its generalisation performance. But its exact impact is still unclear. Therefore, we train neural networks with different architectures on the mountain car policy evaluation problem with different amount of samples, and evaluate their performance with sum of squared TD errors Z on unseen samples in log scale(Figure 2). Here we evaluate the value of a fixed policy, which is to accelerate the car in the direction of the current velocity. The result shows that, a simple network is able to generalise for a continuous state space, while deeper networks are more sensitive to the training set size. Interestingly, if we consider $\log_{10} Z \leq -1$ as the threshold for good performance, the required number of samples to achieve this threshold stays similar (≈ 600) as we vary the depth of the network. In other words, deeper networks do not obviously increase this threshold of required samples in our experiment, they rather *reinforce* the cases where the number of samples is far from this threshold in both directions; a sample size larger than the threshold generalises even better, while a smaller size gets punished harder, when using deeper architectures. This could be an interesting hypothesis to investigate and verify in the future work.

1. In our tests, the gradient only methods require roughly 4.4 seconds on average for 10^4 iterations, while using the Hessian increases the run time to 7.1 seconds. This means within 10^4 steps of first order methods, around $6.2 \cdot 10^3$ iterations of the AN algorithm can be performed. The performance gain in terms of convergence speed and overall lower error, as seen in Figure 1(b) and 1(c), justify the additional computational effort.



(a) $T = 50, i = 5$ (b) $T = 500, i = 5$ (c) $T = 50, i = 25$ (d) $T = 300, i = 25$

(e) $T = 50, i = 5$ (f) $T = 500, i = 5$ (g) $T = 50, i = 25$ (h) $T = 300, i = 25$

Figure 4: The Cartpole experiment: rewards vs. iterations (top: gradient descent, bottom: AN algorithm, ignoring(blue)/considering(red) gradients of TD target).

Off-Policy Policy Evaluation For reinforcement learning algorithms, it is sometimes necessary to evaluate a policy other than the current behavior policy. Here we show how our algorithm can be used for such an off-policy policy evaluation scenario. This experiment is done on the mountain car problem. We apply a stochastic behavior policy μ, where the desired action is executed with a probability of $p = 0.4$ while each of the remaining actions is executed with $p = 0.3$. The target policy to be evaluated is to execute the desired action with $p = 0.8$, and $p = 0.1$ for each of the remaining actions. The off-policy learning is realised by including importance sampling factors $\rho(s, a) = \frac{\pi(s,a)}{\mu(s,a)}$ in front of δ^2 in Eq. (3). Figure 3 shows that V_π approximated by the FNN is getting close to the ground truth solution obtained by Monte Carlo methods. Considering gradients of the TD target together with second order information achieves the best performance. First order gradient descent that ignores the TD target is comparably fast in the initial phase, but it exhibits a more instable behavior over time. We hypothesise that this is because ignoring the TD target provides less accurate update directions when getting closer to the local optimum.

Policy Iteration We test our approach in a policy iteration fashion on the well-known cartpole problem. This is done by using Q-factors in place of $V_\pi(s)$, with an MLP of size $\boldsymbol{\mathcal{F}}(5, 10, 10, 1)$. We define the policy to be ϵ-greedy with $\epsilon = 0.5$. Here, the network input consists of the state and action. We test with different sample sizes $T \in \{50, 300, 500\}$, i.e. the number of actions executed in every iteration, and different policy evaluation steps $i \in \{5, 25\}$, i.e. optimisation steps that are performed for each evaluation. The results show that the AN algorithm requires less policy evaluation steps i to perform well (Figures 4(e), 4(f)), compared to first order approaches (Figures 4(a), 4(b)). However, second order approaches rely on a good estimate of the Hessian, therefore less samples (smaller T) may cause problems, especially for the case with more policy evaluation steps ($i = 25$) (Figure 4(g)). For these cases, first order approaches learn the value function steadily, and hence refrains itself from over-adapting to the incomplete sample set (Figure 4(c)). This problem resolves once enough samples are given to estimate the Hessian more accurately (Figure 4(h)). These results imply that second order information can be helpful to accele-

rate and stabilise policy iteration in continuous problems. It also shows that, in order to get the benefits of the second order approaches, the TD target cannot be ignored, otherwise the policy iteration learning process may collapse (Figures 4(e), 4(g) and 4(h)).

4. Conclusions

This work investigates the problem of minimizing the expected one step TD error for the continuous state space with NN value function approximators, and proposes a second order approximate Newton's algorithm. We reconsider the question of whether the gradients of the TD target should be used, and conduct an extensive comparison. Our results show that ignoring gradients of the TD target works reasonably well with first order approaches, demonstrates good performance in the initial phase, and during the policy iteration process where only very few samples are collected for each evaluation step. On the other hand, the proposed approximate Newton's method is more efficient and offers better convergence, when a well estimated Hessian of the optimisation problem can be given. This can be observed in both policy evaluation and full policy iteration experiments. In general, the results indicate that considering gradients of the TD target can be interesting, especially for second order algorithms for training neural network value function approximators.

References

P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.

L. C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *Proceeding of the* 12^{th} *International Conference on Machine Learning*, pages 30–37, 1995.

D. P. Bertsekas. *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, volume 2. Athena Scientific, 4^{th} edition, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pages 1204–1212, 2009.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Peterson, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, pages 317–328, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29243-8, 978-3-540-29243-2. doi: 10.1007/11564096_32. URL `http://dx.doi.org/10.1007/11564096_32`.

Hao Shen. Towards a mathematical understanding of the difficulty in learning with feedforward neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

D. Silver. Gradient temporal difference networks. In Marc Peter Deisenroth, Csaba Szepesvári, and Jan Peters, editors, *Proceedings of the* 10*-th European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 117–130, 2013.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 550: 354—359, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

R. S. Sutton, Csaba Szepesvári, and H. R. Maei. A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximations. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, volume 21, pages 1609–1616. The MIT Press, 2008.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the* 26^{th} *International Conference on Machine Learning (ICML)*, pages 993–1000, 2009.

D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer-Verlag, London, 2015.