Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning

Stefan Elfwing[a] elfwing@atr.jp
Eiji Uchibe[a,b] uchibe@atr.jp
Kenji Doya[b] doya@oist.jp

[a]Dept. of Brain Robot Interface, ATR Computational Neuroscience Laboratories, 2-2-2 Hikaridai, Seikacho, Soraku-gun, Kyoto 619-0288, Japan
[b]Neural Computation Unit, Okinawa Institute of Science and Technology Graduate University, 1919-1 Tancha, Onna-son, Okinawa 904-0495, Japan

Abstract

In recent years, neural networks have enjoyed a renaissance as function approximators in reinforcement learning. Two decades after Teasauro's TD-Gammon achieved near top-level human performance in backgammon, the deep reinforcement learning algorithm DQN (combining Q-learning with a deep neural network, experience replay, and a separate target network) achieved human-level performance in many Atari 2600 games. The purpose of this study is twofold. First, we propose two activation functions for neural network function approximation in reinforcement learning: the sigmoid-weighted linear (SiL) unit and its derivative function (dSiL). The activation of the SiL unit is computed by the sigmoid function multiplied by its input. Second, we suggest that the more traditional approach of using on-policy learning with eligibility traces, instead of experience replay, and softmax action selection can be competitive with DQN, without the need for a separate target network. We validate our proposed approach by, first, achieving new state-of-the-art results in both stochastic SZ-Tetris and Tetris with a small 10×10 board, using TD(λ) learning and shallow dSiL network agents, and, then, outperforming DQN in the Atari 2600 domain by using a deep Sarsa(λ) agent with SiL and dSiL hidden units.

1 Introduction

Neural networks have enjoyed a renaissance as function approximators in reinforcement learning (Sutton and Barto, 1998) in recent years. The DQN algorithm (Mnih et al., 2015), which combines Q-learning with a deep neural network, experience replay, and a separate target network, achieved human-level performance in many Atari 2600 games. Since the development of the DQN algorithm, there have been several proposed improvements, both to DQN specifically and deep reinforcement learning in general. Van Hasselt et al. (2015) proposed double DQN to reduce overestimation of the action values in DQN and Schaul et al. (2016) developed a framework for more efficient replay by prioritizing experiences of more important state transitions. Wang et al. (2016) proposed the dueling network architecture for more efficient learning of the action value function by separately estimating the state value function and the advantages of each action. Mnih et al. (2016) proposed a framework for asynchronous learning by multiple agents in parallel, both for value-based and actor-critic methods.

The purpose of this study is twofold. First, motivated by the high performance of the expected energy restricted Boltzmann machine (EE-RBM; Elfwing et al., 2015, 2016), we propose two activation functions for neural network function approximation in reinforcement learning: the sigmoid-weighted linear (SiL) unit and its derivative function (dSiL). The activation of the SiL unit is computed by the sigmoid function multiplied by its input and it looks like a continuous and “undershooting” version of the linear rectifier (ReL) unit (Hahnloser et al., 2000). The activation of the dSiL unit looks like steeper and “overshooting” version of the sigmoid function.

Second, we suggest that the more traditional approach of using on-policy learning with eligibility traces, instead of experience replay, and softmax action selection with simple annealing can be competitive with DQN, without the need for a separate target network. Our approach is something of a throwback to the approach used by Tesauro (1994) to develop TD-Gammon more than two decades ago. Using a neural network function approximator and TD(λ) learning (Sutton, 1988), TD-Gammon reached near top-level human performance in backgammon, which to this day remains one of the most impressive application of reinforcement learning.

To evaluate our proposed approach, we first test the performance of shallow network agents with SiL, ReL, dSiL, and sigmoid hidden units in stochastic SZ-Tetris, which is a simplified but difficult version of Tetris. The best agent, the dSiL network agent, improves the average state-of-the-art score by 20 %. We thereafter train a dSiL network agent in standard Tetris with a smaller, 10×10, board size, achieving a state-of-the-art score in this more competitive version of Tetris as well. We then test a deep network agent, with SiL and dSiL hidden units, in the Atari 2600 domain. It improves the mean DQN normalized scores achieved by DQN and double DQN by 232 % and 161 %, respectively, in 12 unbiasedly selected games. We finally analyze the ability of on-policy value-based reinforcement learning to accurate estimate the expected discounted returns and the importance of softmax action selection for the games where our proposed agents performed particularly well.

2 Method

2.1 TD(λ) and Sarsa(λ)

In this study, we use two reinforcement learning algorithms: TD(λ) (Sutton, 1988) and Sarsa(λ) (Rummery and Niranjan, 1994; Sutton, 1996). TD(λ) learns an estimate of the state-value function, V^{π}, and Sarsa(λ) learns an estimate of the action-value function, Q^{π}, while the agent follows policy π. If the approximated value functions, $V_t \approx V^{\pi}$ and $Q_t \approx Q^{\pi}$, are parameterized by the parameter vector $\boldsymbol{\theta}_t$, then the gradient-descent learning update of the parameters is computed by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \delta_t \boldsymbol{e}_t, \tag{1}$$

where the TD-error, δ_t, is

$$\delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{2}$$

for TD(λ) and

$$\delta_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{3}$$

for Sarsa(λ). The eligibility trace vector, $\boldsymbol{e}_t$, is

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} V_t(s_t),\ \boldsymbol{e}_0 = \mathbf{0}, \tag{4}$$

for TD(λ) and

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q_t(s_t, a_t),\ \boldsymbol{e}_0 = \mathbf{0}, \tag{5}$$

for Sarsa(λ). Here, s_t is the state at time t, a_t is the action selected at time t, r_t is the reward for taking action a_t in state s_t, α is the learning rate, γ is the discount factor of future rewards, λ is the trace-decay rate, and $\nabla_{\boldsymbol{\theta}_t} V_t$ and $\nabla_{\boldsymbol{\theta}_t} Q_t$ are the vectors of partial derivatives of the function approximators with respect to each component of $\boldsymbol{\theta}_t$.

2.2 Sigmoid-weighted Linear Units

In our earlier work (Elfwing et al., 2016), we proposed the EE-RBM as a function approximator in reinforcement learning. In the case of state-value based learning, given a state vector $\mathbf{s}$, an EE-RBM approximates the state-value function V by the negative expected energy of an RBM (Smolensky, 1986; Freund and Haussler, 1992; Hinton, 2002) network:

$$V(\mathbf{s}) = \sum_k z_k \sigma(z_k) + \sum_i b_i s_i, \tag{6}$$

$$z_k = \sum_i w_{ik} s_i + b_k, \tag{7}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{8}$$

Here, z_k is the input to hidden unit k, $\sigma(\cdot)$ is the sigmoid function, b_i is the bias weight for input unit s_i, w_{ik} is the weight connecting state s_i and hidden unit k, b_k is the bias weight for hidden unit k. Note that Equation 6 can be regarded as the output of a one-hidden layer feedforward neural network with hidden unit activations computed by $z_k\sigma(z_k)$ and with uniform output weights of one. In this study, motivated by the high performance of the EE-RBM in both the classification (Elfwing et al., 2015) and the reinforcement learning (Elfwing et al., 2016) domains, we propose the SiL unit as an activation function for neural network function approximation in reinforcement learning. The activation a_k of a SiL unit k for an input vector $\mathbf{s}$ is computed by the sigmoid function multiplied by its input:

$$a_k(\mathbf{s}) = z_k \sigma(z_k). \tag{9}$$

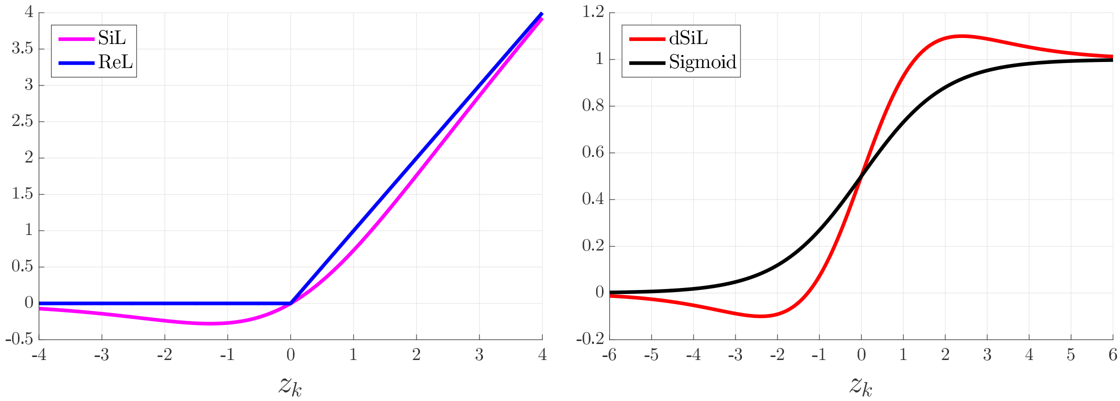

Figure 1: The activation functions of the SiL and ReL units (left panel) and the dSiL and sigmoid units (right panel).

For z_k-values of large magnitude, the activation of the SiL unit is approximately equal to the activation of the ReL unit (see left panel in Figure 1), i.e., the activation is approximately equal to zero for large negative z_k-values and approximately equal to z_k for large positive z_k-values. Unlike the ReL unit (and other commonly used activation units such as sigmoid and tanh units), the activation of the SiL unit is not monotonically increasing. Instead, it has a global minimum value of approximately -0.28 for $z_k \approx -1.28$. An attractive feature of the SiL unit is that it has a self-stabilizing property, which we demonstrated experimentally in Elfwing et al. (2015). The global minimum, where the derivative is zero, functions as a "soft floor" on the weights that serves as an implicit regularizer that inhibits the learning of weights of large magnitudes.

We propose an additional activation function for neural network function approximation: the dSiL unit. The activation of the dSiL unit is computed by the derivative of the SiL unit:

$$a_k(\mathbf{s}) = \sigma(z_k)\left(1 + z_k(1 - \sigma(z_k))\right). \tag{10}$$

The activation function of the dSiL unit looks like an steeper and "overshooting" sigmoid function (see right panel in Figure 1). The dSiL unit has a maximum value of approximately 1.1 and a minimum value of approximately -0.1 for $z_k \approx \pm 2.4$, i.e., the solutions to the equation $z_k = -\log\left((z_k - 2)/(z_k + 2)\right)$.

The derivative of the activation function of the SiL unit, used for gradient-descent learning updates of the neural network weight parameters (see Equations 4 and 5), is given by

$$\nabla_{w_{ik}} a_k(\mathbf{s}) = \sigma(z_k)\left(1 + z_k(1 - \sigma(z_k))\right) s_i, \tag{11}$$

and the derivative of the activation function of the dSiL unit is given by

$$\nabla_{w_{ik}} a_k(\mathbf{s}) = \sigma(z_k)(1 - \sigma(z_k))(2 + z_k(1 - \sigma(z_k)) - z_k\sigma(z_k))s_i. \tag{12}$$

2.3 Action selection

We use softmax action selection with a Boltzmann distribution in all experiments. For Sarsa(λ), the probability to select action a in state s is defined as

$$\pi(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_b \exp(Q(s,b)/\tau)}. \tag{13}$$

For the model-based TD(λ) algorithm, we select an action a in state s that leads to the next state s' with a probability defined as

$$\pi(a|s) = \frac{\exp(V(f(s,a))/\tau)}{\sum_b \exp(V(f(s,b))/\tau)}. \tag{14}$$

Here, $f(s,a)$ returns the next state s' according to the state transition dynamics and τ is the temperature that controls the trade-off between exploration and exploitation. We used hyperbolic annealing of the temperature and the temperature was decreased after every episode i:

$$\tau(i) = \frac{\tau_0}{1 + \tau_k i}. \tag{15}$$

Here, τ_0 is the initial temperature and τ_k controls the rate of annealing.

3 Experiments

3.1 SZ-Tetris

Szita and Szepesvári (2010) proposed stochastic SZ-Tetris (Burgiel, 1997) as a benchmark for reinforcement learning that preserves the core challenges of standard Tetris but allows faster evaluation of different strategies due to shorter episodes by removing easier tetrominos. Stochastic SZ-Tetris is played on a board of standard Tetris size with a width of 10 and a height of 20. In each time step, either an S-shaped tetromino or a Z-shaped tetromino appears with equal probability. The agent selects a rotation (lying or standing) and a horizontal position within the board. In total, there are 17 possible actions for each tetromino (9 standing and 8 lying horizontal positions). After the action selection, the tetromino drops down the board, stopping when it hits another tetromino or the bottom of the board. If a row is completed, then it disappears. The agent gets a score of +1 point for each completed row. An episode ends when a tetromino does not fit within the board.

For an alternating sequence of S-shaped and Z-shaped tetrominos, the upper bound on the episode length in SZ-Tetris is 69 600 fallen pieces (Burgiel, 1997) (corresponding to a score of 27 840 points), but the maximum episode length is probably much shorter, maybe a few thousands (Szita and Szepesvári, 2010). That means that to evaluate a good strategy SZ-Tetris requires at least five orders of magnitude less computation than standard Tetris.

The standard learning approach for Tetris has been to use a model-based setting and define the evaluation function or state-value function as the linear combination of hand-coded features. Value-based reinforcement learning algorithms have a lousy track record using this approach. In regular Tetris, their reported performance levels are many magnitudes lower than black-box methods such as the cross-entropy (CE) method and evolutionary approaches. In stochastic SZ-Tetris, the reported scores for a wide variety of reinforcement learning algorithms are either approximately zero (Szita and Szepesvári, 2010) or in the single digits [1].

Value-based reinforcement learning has had better success in stochastic SZ-Tetris when using non-linear neural network based function approximators. Faußer and Schwenker (2013) achieved a score of about 130 points using a shallow neural network function approximator with sigmoid hidden units. They improved the result to about 150 points by using an ensemble approach consisting of ten neural networks. In our earlier study (Elfwing et al., 2016), we achieved an average score of about 200 points using three different neural network function approximators: an EE-RBM, a free energy RBM, and a standard neural network with sigmoid hidden units. Jaskowski et al. (2015) achieved the current state-of-the-art results using systematic n-tuple networks as function approximators: average scores of 220 and 218 points achieved by the evolutionary VD-CMA-ES method and TD-learning, respectively, and the best mean score in a single run of 295 points achieved by TD-learning.

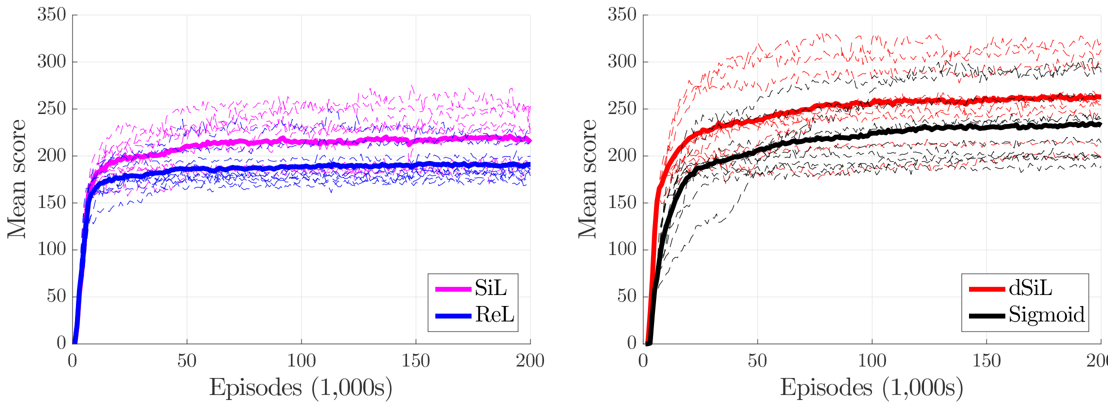

Figure 2: Learning curves in stochastic SZ-Tetris for the four types of neural network agents. The figure shows the average scores over ten separate runs (tick solid lines) and the scores of individual runs (thin dashed lines). The mean scores were computed over every 1,000 episodes.

In this study, we compare the performance of shallow network agents with SiL, ReL, dSiL, and sigmoid hidden units. We used the same experimental setup as in our earlier work (Elfwing et al., 2016). The networks consisted of one hidden layer with 50 hidden units and a linear output layer. The agents were trained by the TD(λ) algorithm and softmax action selection. The features were similar to the original 21 features proposed

[1] http://barbados2011.rl-community.org/program/SzitaTalk.pdf

by Bertsekas and Ioffe (1996), except for not including the maximum column height and using the differences in column heights instead of the absolute differences: 10 column heights (10×21 binary states), 9 column height differences that were capped at ±5 (9×11 binary states), and the number of holes (empty cells covered by at least one occupied cell; 151 binary states, i.e., the number was capped at 150 holes). The length of the binary state vector was 460. We used the following reward function (proposed by Faußer and Schwenker (2013)):

$$r(\boldsymbol{s}) = e^{-(\text{number of holes in } \boldsymbol{s})/33}. \tag{16}$$

We set γ to 0.99, λ to 0.55, τ_0 to 0.5, and τ_k to 0.00025. We used a rough grid-like search for each of the four neural network agents to find appropriate values of the learning rate α and it was determined to be 0.001 for all four agents.

Table 1: Average scores (± standard deviations) achieved in stochastic SZ-Tetris, computed over the final 1,000 episodes for all ten runs and the best individual runs.

Network	**Final average score**	**Final best score**
SiL	214 ± 74	253 ± 83
ReL	191 ± 58	227 ± 76
dSiL	$\mathbf{263 \pm 80}$	$\mathbf{320 \pm 87}$
Sigmoid	232 ± 75	293 ± 73

The agents were trained for 200,000 episodes and the experiments were repeated for ten separate runs for each type of neural network. The average learning curves as well as learning curves for the individual runs are shown in Figure 2. The final results are summarized in Table 1. The results show significant differences ($p < 0.0001$) in final average score between all four agents. The networks with bounded hidden units (dSiL and sigmoid) outperformed the networks with unbounded units (SiL and ReL), the SiL network outperformed the ReL network, and the dSiL network outperformed the sigmoid network. The final average score (best score) of 263 (320) points achieved by the dSiL network agent is a new state-of-the-art score, improving the previous best performance by 43 (25) points or 20 % (8 %). The almost 30 points better final average score achieved by the sigmoid network agent, compared with the result in our earlier study (Elfwing et al., 2016), is explained by a more appropriate setting of the trace-decay rate λ, 0.55 instead of 0.8.

3.2 10×10 Tetris

The result achieved by the dSiL network agent in stochastic SZ-Tetris is impressive, but we cannot compare the result with the methods that have achieved the highest performance levels in standard Tetris, because those methods have not been applied to stochastic SZ-Tetris. Furthermore, it is not feasible to apply our method to Tetris with a standard board height of 20, because of the prohibitively long learning time. The current state-of-the-art for a single run of an algorithm, achieved by the CBMPI algorithm (Gabillon et al., 2013; Scherrer et al., 2015), is a mean score of 51 million cleared lines. However, for the best methods applied to Tetris, there are reported results for a smaller, 10×10, Tetris board, and in this case the learning time for our method is long, but not prohibitively so.

10×10 Tetris is played with the standard seven tetrominos and the numbers of actions are 9 for the block-shaped tetromino, 17 for the S-, Z-, and stick-shaped tetrominos, and 34 for the J-, L- and T-shaped tetrominos. In each time step, the agent gets a score equal to the number of completed rows, with a maximum of +4 points that can only be achieved by the stick-shaped tetromino.

We trained a neural network agent with dSiL units in the hidden layer. To handle the more complex learning task, we increased the number of hidden units to 250 and the number of episodes to 400,000. We repeated the experiment for five separate runs. We used the same 20 state features as in the SZ-Tetris experiment, but the length of the binary state vector was reduced to 260 due to the smaller board size: 10 column heights (10×11 binary states), 9 column height differences that were capped at ±5 (9×11 binary states), and the number of holes (76 binary states, i.e., the number was capped at 75 holes). The reward function was changed as follows for the same reason:

$$r(\boldsymbol{s}) = e^{-(\text{number of holes in } \boldsymbol{s})/(33/2)}. \tag{17}$$

We used the same values of the meta-parameters as in the stochastic SZ-Tetris experiment.

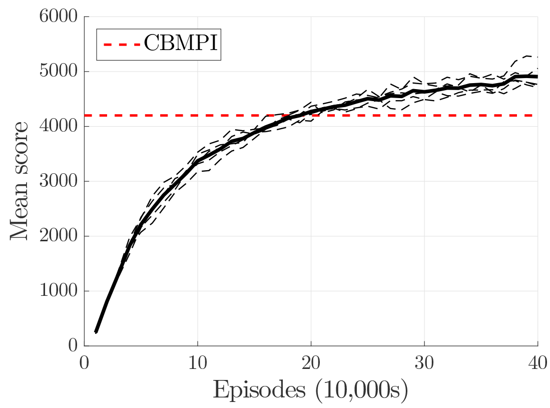

Figure 3: Learning curves for a dSiL network agent with 250 hidden nodes in 10×10 Tetris. The figure shows the average score over five separate runs (tick solid lines) and the scores of individual runs (thin dashed lines). The red dashed line show the previous best average score of 4,200 points achieved by the CBMPI algorithm.

The average learning curve as well as learning curves for the five separate runs in 10×10 Tetris are shown in Figure 3. The dSiL network agent reached an average score of 4,900 points over the final 10,000 episodes and the five separate runs, which is a new state-of-the-art in 10×10 Tetris. The previous best average scores are 4,200 points achieved by the

CBMPI algorithm, 3,400 points achieved by the DPI algorithm, and 3,000 points achieved by the CE method (Gabillon et al., 2013). The best individual run achieved a final mean score of 5,300 points, which is also a new state-of-the-art, improving on the score of 5,000 points achieved by the CBMPI algorithm.

It is particularly impressive that the dSiL network agent achieved its result using features similar to the original Bertsekas features. Using only the Bertsekas features, the CBMPI algorithm, the DPI algorithm, and the CE method could only achieve average scores of about 500 points (Gabillon et al., 2013). The CE method has achieved its best score by combining the Bertsekas features, the Dellacherie features (Fahey, 2003), and three original features (Thiery and Scherrer, 2009). The CBMPI algorithm achieved its best score using the same features as the CE method, except for using five original RBF height features instead of the Bertsekas features.

3.3 Atari 2600 games

To further evaluate the use of value-based on-policy reinforcement learning with eligibility traces and softmax action selection in high-dimensional state space domains, as well as the use of SiL and dSiL units, we applied Sarsa(λ) with a deep convolution neural network function approximator in the Atari 2600 domain using the Arcade Learning Environment (Bellemare et al., 2013). We used SiL units in the convolutional layers, dSiL units in the hidden fully-connected layer, and a linear output layer (hereafter, *deep SiL agent*). To limit the number of games and prevent a biased selection of the games, we selected the 12 games played by DQN (Mnih et al., 2015) that started with the letters 'A' and 'B': Alien, Amidar, Assault, Asterix, Asteroids, Atlantis, Bank Heist, Battle Zone, Beam Rider, Bowling, Boxing, Breakout.

We used a similar experimental setup as Mnih et al. (2015). We pre-processed the raw 210×160 Atari 2600 RGB frames by extracting the luminance channel, taking the maximum pixel values over consecutive frames to prevent flickering, and then downsampling the grayscale images to 105×80. For computational reasons, we used a smaller network architecture. Instead of three convolutional layers, we used two with half the number of filters, each followed by a max-pooling layer. The input to the network was a 105×80×2 image consisting of the current and the fourth previous pre-processed frame. As we used frame skipping where actions were selected every fourth frame and repeated for the next four frames, we only needed to apply pre-processing to every fourth frame. The first convolutional layer had 16 filters of size 8×8 with a stride of 4. The second convolutional layer had 32 filters of size 4×4 with a stride of 2. The max-pooling layers had pooling windows of size 3×3 with a stride of 2. The convolutional layers were followed by a fully-connected hidden layer with 512 dSiL units and a fully-connected linear output layer with 4 to 18 output (or action-value) units, depending on the number of valid actions in the considered game.

We selected meta-parameters by a preliminary search in the Alien, Amidar and Assault games and used the same values for all 12 games: α: 0.001, γ: 0.99, λ: 0.8, τ_0: 0.5, and τ_k: 0.0005. As in Mnih et al. (2015), we clipped the rewards to be between -1 and $+1$, but we did not clip the values of the TD-errors.

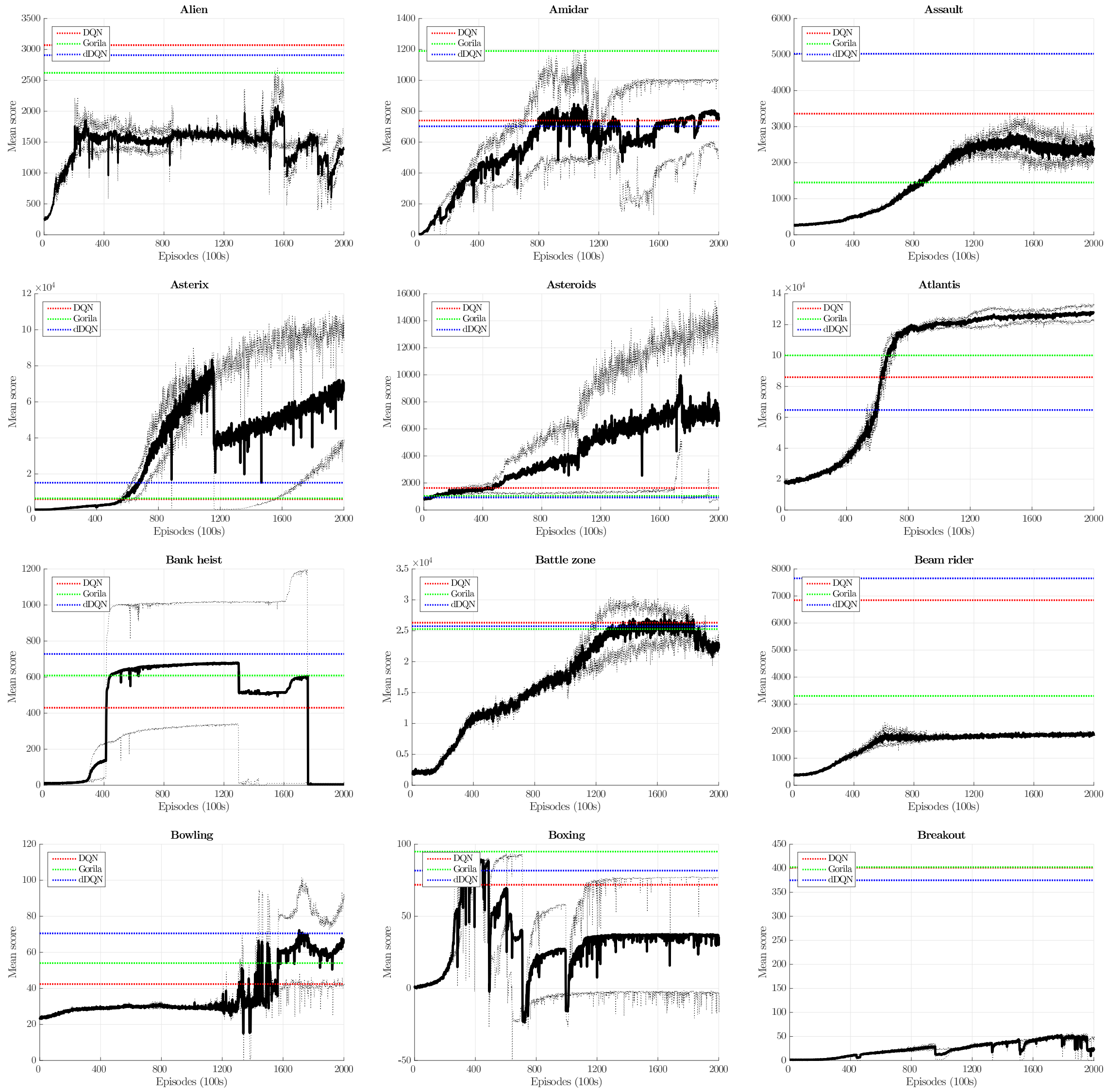

Figure 4: Average learning curves (solid lines) over two separate runs (dashed lines) for the deep SiL agents in the 12 Atari games. The dotted lines show the reported results for DQN (red), the Gorila implementation of DQN (green), and double DQN (blue).

In each of the 12 Atari games, we trained a deep SiL agent for 200,000 episodes and the experiments were repeated for two separate runs. An episode started with up to 30 'do nothing' actions (*no-op condition*) and it was played until the end of the game or for a maximum of 18,000 frames (i.e., 5 minutes). Figure 4 shows the average learning curves, as well as the learning curves for the two separate runs, in the 12 Atari 2600 games. Table 2 summarizes the results as the *final mean scores* computed over the final 100 episodes for the two separate runs, and the *best mean scores* computed as average scores of the highest mean scores (over every 100 episodes) achieved in each of the two runs. The table also shows the reported best mean scores for single runs of DQN computed over 30 episodes, average scores over five separate runs of the Gorila implementation of DQN (Nair et al., 2015) computed over 30 episodes, and single runs of double DQN (van Hasselt et al., 2015) computed over 100 episodes. The last two rows of the table shows summary statistics over the 12 games, which were obtained by computing the mean and the median of the DQN normalized scores:

$$\text{Score}_{\text{DQN\_normalized}} = \frac{\text{Score}_{\text{agent}} - \text{Score}_{\text{random}}}{\text{Score}_{\text{DQN}} - \text{Score}_{\text{random}}}$$

Here, $\text{Score}_{\text{random}}$ is the score achieved by a random agent in Mnih et al. (2015).

Table 2: The final and best mean scores achieved by the deep SiL agents in 12 Atari 2600 games, and the reported best mean scores achieved by DQN, the Gorila implementation of DQN, and double DQN in the no-op condition with 5 minutes of evaluation time.

				deep SiL	
Game	**DQN**	**Gorila**	**double DQN**	**Final**	**Best**
Alien	**3,069**	2,621	2,907	1,370	2,246
Amidar	740	**1,190**	702	762	904
Assault	3,359	1,450	**5,023**	2,415	2,944
Asterix	6,012	6,433	15,150	70,942	**100,322**
Asteroids	1,629	1,048	931	6,537	**10,614**
Atlantis	85,950	100,069	64,758	127,651	**128,983**
Bank Heist	430	609	728	5	**770**
Battle Zone	26,300	25,267	25,730	22,930	**29,115**
Beam Rider	6,846	3,303	**7,654**	1,829	2,176
Bowling	42	54	71	67	**75**
Boxing	72	**95**	82	36	92
Breakout	401	**402**	375	25	55
Mean (DQN Normalized)	100 %	102 %	127 %	218 %	**332** %
Median (DQN Normalized)	100 %	104 %	105 %	78 %	**125** %

The results clearly show that our deep SiL agent outperformed the other agents, improving the mean (median) DQN normalized best mean score from 127 % (105 %) achieved by double DQN to 332 % (125 %). The deep SiL agent achieved the highest best mean score in 6 out of the 12 games and only performed much worse than the other 3 agents in one game, Breakout, where the learning never took off during the 200,000 episodes of training (see Figure 4). The performance was especially impressive in the Asterix (score of 100,322) and Asteroids (score of 10,614) games, which improved the best mean performance achieved by the second-best agent by 562 % and 552 %, respectively.

4 Analysis

4.1 Value estimation

First, we investigate the ability of TD(λ) and Sarsa(λ) to accurately estimate discounted returns:

$$R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}.$$

Here T is the length of an episode. The reason for doing this is that van Hasselt et al. (2015) showed that the double DQN algorithm improved the performance of DQN in Atari 2600 games by reducing the overestimation of the action values. It is known (Thrun and Schwartz, 1993; van Hasselt, 2010) that Q-learning based algorithms, such as DQN, can overestimate action values due to the max operator, which is used in the computation of the learning targets. TD(λ) and Sarsa(λ) do not use the max operator to compute the learning targets and they should therefore not suffer from this problem.

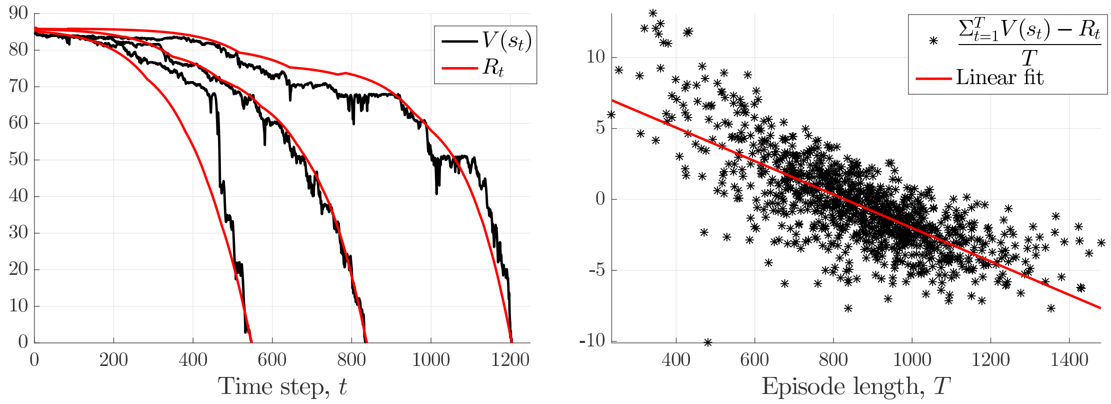

Figure 5: The left panel shows learned $V(s_t)$-values and R_t-values, for examples of short, medium-long, and long episodes in SZ-Tetris. The right panel shows the normalized sum of differences between $V(s_t)$ and R_t for 1,000 episodes and the best linear fit of the data ($-0.012T + 9.8$).

Figure 5 shows that for episodes of average (or expected) length the best dSiL network agent in SZ-Tetris learned good estimates of the discounted returns, both along the episodes (left panel) and as measured by the normalized sum of differences between $V(s_t)$ and R_t (right panel):

$$\frac{1}{T} \sum_{t=1}^{T} \left(V(s_t) - R_t \right).$$

The linear fit of the normalized sum of differences data for 1,000 episodes gives a small underestimation (-0.43) for an episode of average length (866 time steps). The $V(s_t)$-values overestimated the discounted returns for short episodes and underestimated the discounted returns for long episodes (especially in the middle part of the episodes), which is accurate since the episodes ended earlier and later, respectively, than were expected.

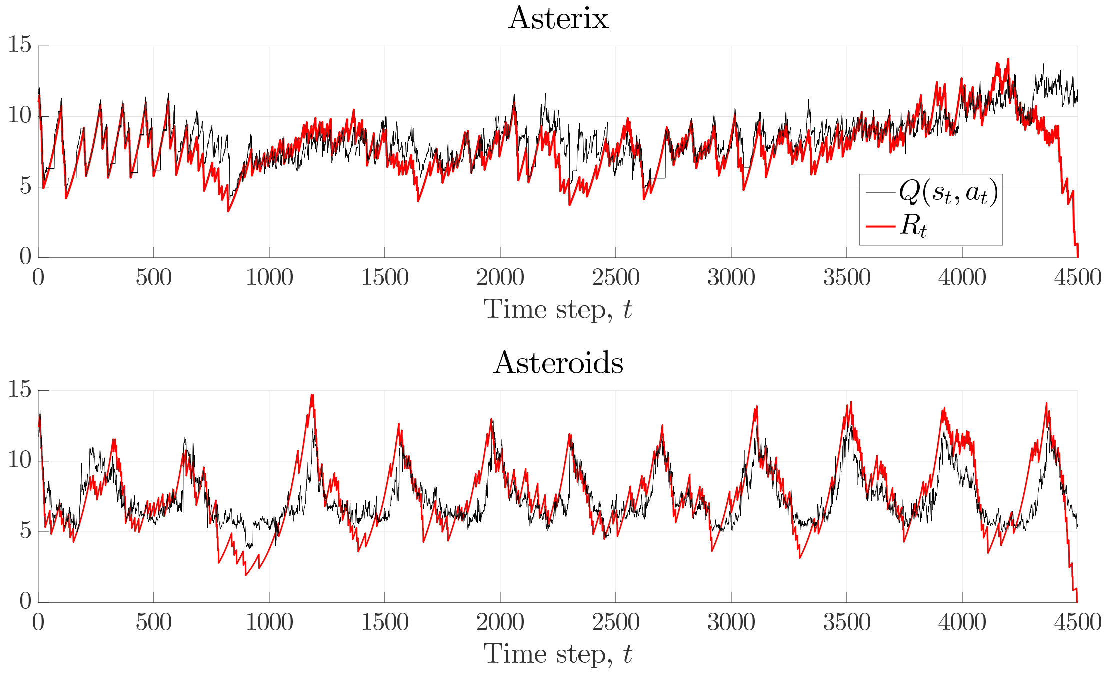

Figure 6: Learned action values, $Q(s_t, a_t)$, and discounted returns, R_t, for the best deep SiL agents in Asterix and Asteroids.

Figure 6 shows typical examples of learned action values and discounted returns along episodes where the best deep SiL agents in Asterix (score of 108,500) and Asteroids (score of 22,500) successfully played for the full 18,000 frames (i.e., 4,500 time steps since the agents acted every fourth frame). In both games, with the exception of a few smaller parts, the learned action values matched the discounted returns very well along the whole episodes. The normalized sums of differences (absolute differences) were 0.59 (1.05) in the Asterix episode and −0.23 (1.28) in the Asteroids episode. In both games, the agents overestimated action values at the end of the episodes. However, this is an artifact of that an episode ended after a maximum of 4,500 time steps, which the agents could not predict. Videos of the corresponding learned behaviors in Asterix and Asteroids can be found at `http://www.cns.atr.jp/~elfwing/videos/asterix_deep_SiL.mov` and `http://www.cns.atr.jp/~elfwing/videos/asteroids_deep_SiL.mov`.

4.2 Action selection

Second, we investigate the importance of softmax action selection in the games where our proposed agents performed particularly well. Almost all deep reinforcement learning algorithms that have been used in the Atari 2600 domain have used ε-greedy action selection (one exception is the asynchronous advantage actor-critic method, A3C, which used softmax output units for the actor (Mnih et al., 2016)). One drawback of ε-greedy selection is that it selects all actions with equal probability when exploring, which can lead to poor learning outcomes in tasks where the worst actions have very bad consequences. This is clearly the case in both Tetris games and in the Asterix and Asteroids games. In each state in Tetris, many, and often most, actions will creates holes, which are difficult (especially in SZ-Tetris) to remove. In the Asterix game, random exploratory actions can kill Asterix if executed when Cacofonix's deadly lyres are passing. In the Asteroids game, one of the actions sends the spaceship into hyperspace and makes it reappear in a random location, which has the risk of the spaceship self-destructing or of destroying it by appearing on top of an asteroid.

Table 3: Mean scores and average numbers of exploratory actions for softmax action selection and ε-greedy action selection with ε set to 0, 0.001, 0.01, and 0.05.

Game	**Selection**	**Mean Score**	**Exploratory actions**
SZ-Tetris	$\tau = 0.0098$	326	28.7
	$\epsilon = 0$	332	0
	$\varepsilon = 0.001$	260	0.59
	$\varepsilon = 0.01$	71	2.0
	$\varepsilon = 0.05$	14	3.2
Asterix	$\tau = 0.00495$	104,299	47.6
	$\epsilon = 0$	102,890	0
	$\varepsilon = 0.001$	98,264	3.6
	$\varepsilon = 0.01$	66,113	30.0
	$\varepsilon = 0.05$	7,152	56.8
Asteroids	$\tau = 0.00495$	15,833	31.3
	$\epsilon = 0$	15,091	0
	$\varepsilon = 0.001$	11,105	2.1
	$\varepsilon = 0.01$	3,536	11.7
	$\varepsilon = 0.05$	1,521	47.3

We compared softmax action selection (τ set to the final values) and ε-greedy action selection with ε set to 0, 0.001, 0.01, and 0.05 for the best dSiL network agent in SZ-Tetris and the best deep SiL agents in the Asterix and Asteroids games. The results (see Table 3) clearly show that ε-greedy action selection with ε set to 0.05, as used for evaluation by DQN, is not suitable for these games. The scores were only 4 % to 10 % of the scores for softmax selection. The negative effects of random exploration were largest in Asteroid and SZ-Tetris. Even when ε was set as low as 0.001 and the agent performed only 2.1 exploratory actions per episode in Asteroids and 0.59 in SZ-Tetris, the mean scores were reduced by 30 % and

20 % (26 % and 22 %), respectively, compared with softmax selection ($\varepsilon = 0$).

5 Conclusions

In this study, we proposed the SiL and dSiL units as activation functions for neural network function approximation in reinforcement learning. We demonstrated in stochastic SZ-Tetris that SiL hidden units significantly outperformed ReL hidden units, and dSiL hidden units significantly outperformed sigmoid hidden units. The best agent, the dSiL network agent, achieved a new state-of-the-art in stochastic SZ-Tetris and in 10×10 Tetris. In the Atari 2600 domain, a deep Sarsa(λ) agent with SiL units in the convolutional layers and dSiL units in the fully-connected hidden layer outperformed DQN and double DQN, as measured by mean and median DQN normalized scores.

An additional purpose of this study was to demonstrate that a more traditional approach of using on-policy learning with eligibility traces and softmax selection (i.e., basically a "textbook" version of a reinforcement learning agent but with non-linear neural network function approximators) can be competitive with the approach used by DQN. This means that there is a lot of room for improvements, by, e.g., using, as DQN, a separate target network, but also by using more recent advances such as the dueling architecture (Wang et al., 2016) for more accurate estimates of the action values and asynchronous learning by multiple agents in parallel (Mnih et al., 2016).

Acknowledgments

This work was supported by the project commissioned by the New Energy and Industrial Technology Development Organization (NEDO), JSPS KAKENHI grant 16K12504, and Okinawa Institute of Science and Technology Graduate University research support to KD.

References

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bertsekas, D. P. and Ioffe, S. (1996). Temporal differences based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, MIT.

Burgiel, H. (1997). How to lose at Tetris. *Mathematical Gazette*, 81:194–200.

Elfwing, S., Uchibe, E., and Doya, K. (2015). Expected energy-based restricted boltzmann machine for classification. *Neural Networks*, 64(3):29–38.

Elfwing, S., Uchibe, E., and Doya, K. (2016). From free energy to expected energy: Improving energy-based value function approximation in reinforcement learning. *Neural Networks*, 84:17–27.

Fahey, C. (2003). Tetris AI, computer plays tetris. `colinfahey.com/tetris/tetris.html` [Online; accessed 22-February-2017].

Faußer, S. and Schwenker, F. (2013). Neural network ensembles in reinforcement learning. *Neural Processing Letters*, pages 1–15.

Freund, Y. and Haussler, D. (1992). Unsupervised learning of distributions on binary vectors using two layer networks. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Proceedings of Advances in Neural Information Processing Systems (NIPS1992)*. Morgan Kaufmann.

Gabillon, V., Ghavamzadeh, M., and Scherrer, B. (2013). Approximate dynamic programming finally performs well in the game of tetris. In *Proceedings of Advances in Neural Information Processing Systems (NIPS2013)*, pages 1754–1762.

Hahnloser, R. H. R., Sarpeshka, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–951.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 12(8):1771–1800.

Jaskowski, W., Szubert, M. G., Liskowski, P., and Krawiec, K. (2015). High-dimensional function approximation for knowledge-free reinforcement learning: a case study in SZ-Tetris. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2015)*, pages 567–573.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML2016)*, pages 1928–1937.

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., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A. D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., and Silver, D. (2015). Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations (ICLR2016)*, Puerto Rico.

Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M. (2015). Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16:1629–1676.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* MIT Press.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceedings of Advances in Neural Information Processing Systems (NIPS1996)*, pages 1038–1044. MIT Press.

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

Szita, I. and Szepesvári, C. (2010). SZ-Tetris as a benchmark for studying key problems of reinforcement learning. In *ICML 2010 workshop on machine learning and games.*

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

Thiery, C. and Scherrer, B. (2009). Improvements on learning tetris with cross entropy. *International Computer Games Association Journal*, 32.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263.

van Hasselt, H. (2010). Double q-learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS2010)*, pages 2613–2621.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (NIPS2016)*, pages 1995–2003.