Improving the Diversity of Bootstrapped DQN by Replacing Priors With Noise

Li Meng, Morten Goodwin, Anis Yazidi, *Senior Member, IEEE*, and Paal E. Engelstad

***Abstract*—Q-learning is one of the most well-known reinforcement learning algorithms. There have been tremendous efforts to develop this algorithm using neural networks. Bootstrapped deep *Q*-learning network is amongst them. It utilizes multiple neural network heads to introduce diversity into *Q*-learning. Diversity can sometimes be viewed as the amount of reasonable moves an agent can take at a given state, analogous to the definition of the exploration ratio in RL. Thus, the performance of bootstrapped deep *Q*-learning network is deeply connected with the level of diversity within the algorithm. In the original research, it was pointed out that a random prior could improve the performance of the model. In this article, we further explore the possibility of replacing priors with noise and sample the noise from a Gaussian distribution to introduce more diversity into this algorithm. We conduct our experiment on the Atari benchmark and compare our algorithm to both the original and other related algorithms. The results show that our modification of the bootstrapped deep *Q*-learning algorithm achieves significantly higher evaluation scores across different types of Atari games. Thus, we conclude that replacing priors with noise can improve bootstrapped deep *Q*-learning's performance by ensuring the integrity of diversities.**

***Index Terms*—Atari, convolutional neural networks (CNNs), deep learning (DL), machine learning, reinforcement learning (RL).**

I. Introduction

REINFORCEMENT learning (RL) is a subfield of artificial intelligence (AI) that studies intelligent behaviors through indicative reward signals. The term "intelligence" can be interpreted in different ways. There were efforts toward describing intelligence in mathematical formulas [16]; yet, there is no strictly definable quantitative measure about the level of intelligence that one agent possesses. Intelligence, given its multipronged nature, can involve a variety of skills that we consider as intelligent behaviors. Thus, focusing on one particular intelligent aspect of an agent can be a progressive approach to research AI [28].

RL is suitable in studying games because it requires the input of reward signals, and games typically have well-defined reward mechanisms to guide a player toward goals. On the other hand, it is often challenging to build a simulator that gives adequate reward signals and precisely describes the complexity of the real world. Games also have finite action and time sequences, enabling the quantitative measure of the intelligence that would not be applicable otherwise.

There are some interests in RL to tackle those issues and to apply RL methods on real-world tasks. Inverse RL lets an agent approximate a cost function from the demonstrations [20], whereas imitation learning can imitate the behaviors from demonstrations in an end-to-end approach [12], [13]. Hierarchical RL decomposes a Markov decision process (MDP) into a hierarchy of smaller MDPs, together with the decomposition of corresponding value functions [4]. Those algorithms are valuable for the real-world applications of RL, while they are not irrelevant to traditional RL methods and still use algorithms developed in game environments.

An agent needs to balance the rate of exploration and exploitation in order to maximize the obtained rewards during an RL procedure. The experienced outcomes for an action at a certain state from the exploration phase can be preserved in a value-based manner, which are called Q-values.

There are two types of RL algorithms, distinguished by how they use the behaviour policy and target policy. On-policy algorithms evaluate and improve the same policy that is used to make moves. In other words, their behavior policy and target policy are the same. Q-learning is an off-policy RL method that adopts the different behavior policy and target policy, which stores and updates Q-values based on the Bellman optimality equation by (1) [35]. Here, γ is the discounting factor, $Q^*(s,a)$ is the Q-value at state s with action a, and r_{t+1} is the result value obtained by advancing to state s_{t+1} at time instant $t+1$. Q-learning is off-policy because max chooses a greedy action but not necessarily the actual action the agent takes at s_{t+1}

$$Q^*(s,a) = E\{r_{t+1} + \gamma \text{max}_{a'} Q^*(s_{t+1}, a' | s_t = t, a_t = a)\}. \tag{1}$$

As RL continuously evolves, neural networks (NNs), including convolutional neural networks (CNNs), have been adopted to assist with the updating and exploring routines of RL. Deep reinforcement learning (DRL) is a combination of RL and deep learning (DL) architectures. Deep Q-learning network (DQN)

Manuscript received 2 March 2022; revised 19 May 2022; accepted 12 June 2022. Date of publication 22 June 2022; date of current version 15 December 2023. This work was support by the Research Council of Norway under Contract 270053 from the Experimental Infrastructure for Exploration of Exascale Computing (eX3). *(Corresponding author: Li Meng.)*

Li Meng and Paal E. Engelstad are with the University of Oslo, 0316 Oslo, Norway (e-mail: li.meng@its.uio.no; paal.engelstad@its.uio.no).

Morten Goodwin is with the Centre for Artificial Intelligence Research, University of Agder, 4630 Kristiansand, Norway, and also with the Oslo Metropolitan University, 0176 Oslo, Norway (e-mail: morten.goodwin@uia.no).

Anis Yazidi is with the Oslo Metropolitan University, 0176 Oslo, Norway, with the Norwegian University of Science and Technology, 7034 Trondheim, Norway, and also with the Oslo University Hospital, 0450 Oslo, Norway (e-mail: anisy@oslomet.no).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TG.2022.3185330.

Digital Object Identifier 10.1109/TG.2022.3185330

is one of the DRL methods that record Q-values in NNs instead of the traditional Q-tables. It was designed to play on the Atari games and achieved better than average human scores through feeding their CNN with raw image pixels of games as input [18].

One of the main problems of Q-learning is overestimation because the max operator is repeatedly used during the training. The max operator perpetuates the approximation errors in one direction, i.e., overestimation but not underestimation. Overestimation bias can often contribute to suboptimal performances of Q-learning [31] and lead the model to learn from repetitive moves that can destabilize the weights in the model. Double Q-learning [10] is one of the most successful proposed methods to alleviate the overestimation bias problem in Q-learning. Double Q-learning maintains two separate Q-value functions but only updates one of them according to either (2) or (3). The learning rate is $\alpha(s, a)$, and a^* is $\text{argmax}_a Q^A_\theta(s,' a)$, and b^* is $\text{argmax}_a Q^B_\theta(s,' a)$ at the next state s'. The complete proof of convergence of double Q-learning under similar conditions of Q-learning is given by [10, Th. 1]

$$Q^A(s,a) = Q^A(s,a) + \alpha(s,a)(r + \gamma Q^B(s,' a^*) - Q^A(s,a)) \tag{2}$$

$$Q^B(s,a) = Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s,' b^*) - Q^B(s,a)). \tag{3}$$

Double Q-learning is found to be unequivocally useful when it is integrated with NNs in DRLs. It has become a widely used technique in DRL implementations. Moreover, the convention is that the target network is kept as a copy of the policy Q-network instead of as a separate network in practice, designed to reduce the amount of computation overhead [33]. The amount of steps to synchronize those two NNs is a hyper-parameter to be tuned.

In the meantime, double Q-learning introduces a new problem called underestimation. Underestimation occurs because the target approximation is a weighted estimate of unbiased expected values, which are lower or equal to the maximum expected values [10]. Although underestimation is often considered harmless in most algorithms, it may still introduce certain undesired effects [3].

Similarly, bootstrapped deep Q-learning network (bootstrapped DQN) [22] is one of the methods that aspire to boost the performance of DQN through some simple but efficient modifications. Bootstrapped DQN intends to ameliorate the limited exploration capabilities of Q-learning by enabling deep exploration in complex RL tasks. Randomized value function methods which can be implemented similarly to Thompson sampling [23], [25], [27], [30] share some common traits with bootstrapped DQN in regard to uncertainty estimation. Nevertheless, previous deep exploration methods either only work on MDPs with small finite state spaces or rely on computationally intractable planning [9], [14].

The importance of diversity in RL has been stressed and discussed in [7], where the author replaces the reward function with a surrogate reward which is based on the entropy and mutual information terms. Each skill is sampled from a prior distribution $p(z)$ and should select trajectories as diversified as possible after the training. Their methods have shown promises in solving simple motor control tasks and indicate that diversity can be an important consideration when designing RL algorithms.

Previous research has also explored the effects of noise on RL [8], [24], [26], [32] and, in particular, a method that introduced a prior network to improve the diversity of bootstrapped DQN [21], where an interesting way of integrating priors to bootstrapped DQN was through simple additions of an untrainable prior network to each ensemble member. However, it was observed afterwards that this way of adding priors to bootstrapped DQN does not improve the performance over bootstrapped DQN [32].

There are also many other techniques and algorithms that improve the performance of DQN, such as dueling Q-learning [34]. However, they are orthogonal to the original study of bootstrapped DQN.

II. Method

In DQN, there is typically only one output head, but bootstrapped DQN consists of multiple (K) output heads. This major modification allows bootstrapped DQN to make actions based on a particular head during exploration, yet to utilize an ensemble of votes during evaluation. A randomly preselected head is to be used for each game episode in the exploration phase of bootstrapped DQN. During the updating phase, there are head masks generated from a Bernoulli distribution to decide whether or not the heads should be updated given the currently drawn replay sample. Bernoulli probability is a value in [0,1] to determine how often 1 is drawn from $\{0,1\}$. Increasing the value of Bernoulli probability equates to more heads being trained on the same batches of samples, which, in turn, reduces the diversity of bootstrapped DQN. Surprisingly, a large value for the Bernoulli probability would not degenerate the performance of the algorithm. The masks were found out best to be 1 or a value close to 1, which can accelerate the training speed with little influences on performances compared to when they are set as smaller values.

The benefit of keeping multiple heads is that there is a diversity of network weights among different heads. In DRL implementations, NNs are typically trained more than millions of times in the learning process, in which identical samples can be drawn repetitively from the replay memory in spite of their distinguished individual Bernoulli probability values in bootstrapped DQN. As a drawback of this type of intensive training, smaller values of masks in bootstrapped DQN not only cannot improve the performance of the model but also decelerate the training and the convergence [22]. It is also obvious that the diversity introduced by using K heads will eventually be negligible after they are trained on the same subset of data frequently and there is no new trajectories to be successfully discovered.

Bootstrapped DQN treats the randomly initialized network weights as a prior. However, this random prior cannot assure the diversity after a decent amount of updates. Bootstrapped DQN also uses a shared network body to learn a joint feature representation, so that the predictions of all heads will eventually become similar or even the same to each other. In the search

space that has already been visited thoroughly, different heads would be more likely to converge into the same policy, and, thus, the rate of exploration will be low [5]. A high level of diversity amongst heads is only guaranteed at the states that have never been encountered or been rarely drawn from the replay memory.

Combining this interpretation with the observation from [32], the same conclusion can be drawn that the bootstrapped DQN plus prior method from [21] still suffers from the same degraded diversity problem at the late stage where the model has been trained many times. As a side-effect of using fixed randomized priors, the benefit of diversity only exists in the states that have been trained infrequently and then will be overshadowed by the tremendous amount of updates.

Although we agree on the principles of using randomized priors in [21], we argue that a successful method to combine priors should satisfy the following requirements.

1) Priors have to be independently drawn each time of the NN update, to maintain the level of diversity at the late stage of training.
2) The method should not strengthen the overestimation or the underestimation problem, in other words, not collapsing the model.
3) Priors should still play a role significantly enough to result in more efficient exploration and optimum convergence despite the above point.
4) The method to obtain priors should not be computationally costly.

Thus, our hypothesis is that noise is more suited than a fixed prior in order to satisfy all those requirements. We propose our algorithm based on bootstrapped DQN, which removes the role of priors and introduces more diversity into the algorithm. Replacing priors with noise can introduce diversity by slightly shifting the target value even on a state that has already been visited frequently. In turn, they expand the search horizon of bootstrapped DQN and can help avoid being stuck in a suboptimal solution for a long period at the late training stage.

Our proposal of modification is to combine the noise with target values during the network update. In each update, a distinct noise value is uniquely sampled, scaled, and then added to the target value. This modification requires less adjustments from bootstrapped DQN comparing to changing the parameters of the NN instead.

In order to minimize the amount of computations needed to generate noise at each update, we choose to use a Gaussian distribution $G(\mu, \sigma)$ for sampling noise, which is considerably computationally more efficient than generating a noise network in each update or changing the structure of NNs. Directly operating noise on target values also simplifies the path to find a suitable scaling parameter because the Q-values themselves can serve as anchor points. A larger noise value is desired as the Q-values grow along the training process. The absolute values of noise are, therefore, scaled in accordance with the actual Q-values to be compatible with games having large score values. The parameter $scale$ defined in (4) linearly increases our noise values. Here, β is a small positive parameter

$$scale = 1 + \beta * \max_a Q_\theta^A(s, a). \tag{4}$$

From this equation, it can be seen that $scale$ approximates 1 when $\beta * \max_a Q_\theta^A(s, a)$ is small enough but remains positive. Although there is a risk that the algorithm might exhibit some unstable behaviors at the beginning of training due to small target Q-values oscillating by relatively large noise values, we still consider as necessary to have such a lower bound of the noise. As mentioned above, double Q-learning has the underestimation problem and could potentially yield extremely low Q-value predictions even at the late phase of training. A linear noise without a lower bound might scale to a value that is too small to be any significant in this case. Hence, a lower bound of the noise helps bootstrapped DQN escape the local optimum and explore more efficiently, whereas the initial dithering can be obscured by a large ϵ at the beginning of training where the agent mostly chooses random actions.

The Q-values are typically lower than the maximum expected value due to the underestimation of double Q-learning. However, this underestimating behavior is susceptible to the influences of the choice of β in $scale$. A Gaussian distribution that only generates negative values could oppress the exploring behavior of the agent and hence leads to bad convergence. On the contrary, the underestimating could potentially turn into overestimating if the distribution coincidentally produces any positive values because the intensive NN updates in RL can lead to overestimating even with a trivial positive bias. Thus, β should be designed as a relatively small (positive) value in our algorithm.

Considering the case where $\max_a Q_\theta^A(s, a)$ is nonpositive and then $scale \leq 1$, larger $\max_a Q_\theta^A(s, a)$ would still result in a larger $scale$ as it previously does. Thus, there is no special treatment required for nonpositive Q-values.

Our pseudocode is shown in Algorithm 1. In our algorithm, r is the reward, γ is the discounting factor, ϵ is the exploration ratio, $maxFrames$ is the maximal number of total frames, and $L()$ is the loss function, i.e., smooth L1 loss in this case. M is a Bernoulli distribution and G is a Gaussian distribution, from which we draw m and np, respectively. K is the number of network heads.

We have two NNs. Q_θ^A is the policy network and Q_θ^B is the target network. Q_θ^B is a duplicate of Q_θ^A and they synchronize at an interval of $sync$ frames. For the update, a^* is $\text{argmax}_a Q_{\theta k}^A(s,' a)$, s is the current state, s' is the next state, a is the action, and $terminal$ is a flag indicating whether the game ends or not.

III. Experimental Details

Our experiments are conducted across 49 Atari games on arcade learning environment (ALE), the same as bootstrapped DQN.[1] For a more detailed ablation study on the effects of using noise, we include evaluation curves of ten games out of them. The hyper-parameters and NN architectures are also kept mostly the same to those stated in bootstrapped DQN. The NN architecture is shown in Fig. 1. The list of chosen parameters is shown in Table I.

Adam is an optimization algorithm that uses the adaptive estimates of lower order moments of the gradient [15], whereas

[1][Online]. Available: https://github.com/mengli11235/Bootstrapped-DQN-with-NP

Algorithm 1: Bootstrapped DQN With Priors Replaced by Noise.

Input: ϵ, γ, $maxFrames$, $sync$, M, $L()$, G, K
Parameter: D, Q_θ^A, Q_θ^B
Output: Q_θ^A

$frames \leftarrow 0$
while $frames < maxFrames$ **do**
  Initialize the environment
  Pick a head k uniformly from $\{0, \ldots, K-1\}$ to make actions
  whileGame not finished**do**
    $frames$ += 4
    if $random() < \epsilon$ **then**
      Choose action randomly
    else
      Choose action based on $Q_{\theta k}^A$
    end if
    Store $s, a, s,' r, terminal$ into replay buffer D
    Store sampled random masks m from M into replay buffer D
    Draw minibatch of $s, a, s,' r, terminal, m$ from D
    Update Q_θ^A according to Algorithm 2
  end while
  if $frames\%sync == 0$**then**
    $Q_\theta^B \leftarrow Q_\theta^A$
  end if
end while
return Q_θ^A

Algorithm 2: Network Update.

Input: $s, a, s,' r, terminal, m, L(), G, K$
Parameter: $Q_\theta^A, Q_\theta^B, scale$
Output:

Generate noise np from the Gaussian distribution G
$totalLoss = 0$
for k in $\{0, \ldots, K-1\}$**do**
  $target = r + \gamma Q_{\theta k}^B(s,' a^*) * (1 - terminal) + scale * np_k$
  Compute $loss$ by $L(Q_{\theta k}^A(s, a), target)$
  Mask $loss$ with m
  $totalLoss$ += $loss$
end for
$totalLoss$ /= K
Update Q_θ^A by $totalLoss$ with gradient descent
return

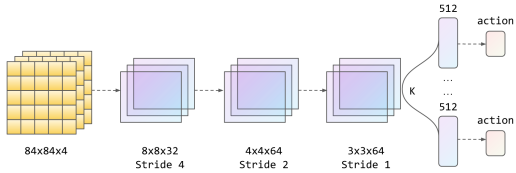


Fig. 1. Structure of our NN. It takes the same input and has the same parameters of convolutional layers as in DQN, and then branches into *K* heads. Each head contains a fully connected layer, followed by an output layer. (a) Scores: Alien. (b) Scores: Amidar. (c) Scores: Assault. (d) Scores: Asterix. (e) Scores: Asteroids. (f) Scores: Atlantis. (g) Scores: Breakout. (h) Scores: Freeway. (i) Scores: Space Invaders. (j) Scores: Tennis.

TABLE I
PARAMETERS USED IN OUR EXPERIMENT

Input	$84 \times 84 \times 4$
k	9
Bernoulli probability	0.9
Optimizer	Adam
Adam learning rate	0.0000625
γ	0.99
Initial ϵ	1
Final ϵ	0.01
ϵ decay frames	1M
$sync$	40000
Frames per step	4
Steps per evaluation	250000
MaxFrames	200M
Replay size	1M
Batch size	32
Noise μ	0
Noise σ	0.02
Noise β	0.05

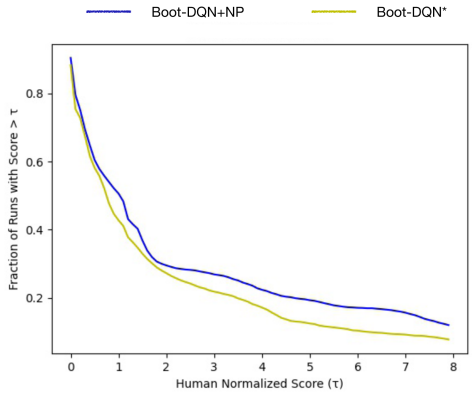


Fig. 2. Performance profiles of 49 Atari games up to $\tau = 8$. The blue line is the percentage of the score larger than τ using noise. The orange line is the percentage of that without using noise.

RMSProp computes the momentum based on rescaled gradients. Although most hyper-parameters followed the chosen values of bootstrapped DQN, we use Adam instead of RMSProp as our optimizer because Adam shows more stable performance on bootstrapped DQN in our experiment. For similar reasons, the value of K is set to 9 instead of 10 with the Bernoulli probability decreased from 1 to 0.9. The hyper-parameter choices of noise-related values are found out through grid search. However, different values of β are found to have similar effects as long as it is set around 0.05.

In most of the Atari games, small rewards were commonly designed to lead to an optimal policy. Thus, the rewards are clipped between –1 and 1 in our experiment to improve the stability. Meanwhile, the reported scores are still raw scores in our results section.

We also use a random no-operation (no-op) period of [0,30] applied at the beginning of each game episode in order to

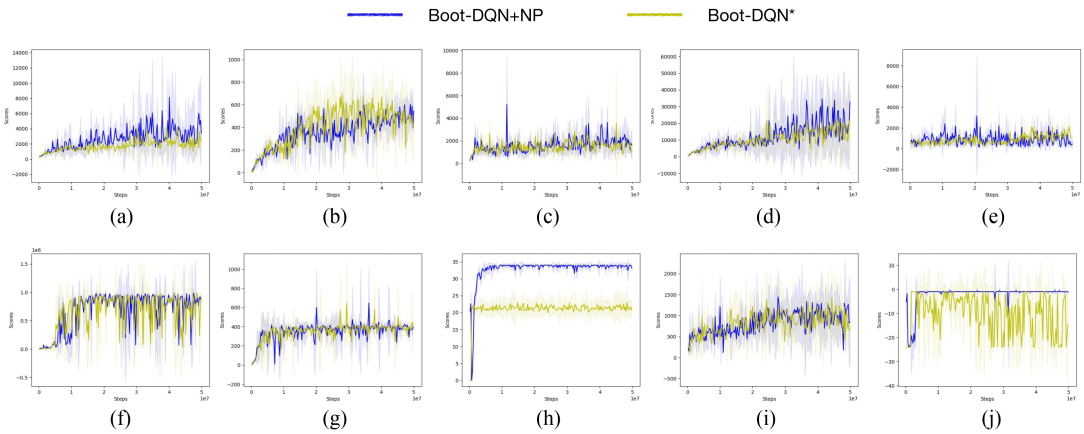


Fig. 3. Evaluation results of playing ten Atari games. The blue lines are the mean evaluation scores of using noise. The orange lines are the mean evaluation scores without using noise. The 95% confidence intervals are also plotted in addition to the scores. All are evaluated every 250 000 steps.

introduce stochasticity into the environment. The seed of the Atari environment is fixed during the training phase, but randomly selected seeds are used for the evaluation process. We run five episodes with those random seeds for each instance of the evaluation phase.

IV. Results

We present our evaluation curves in terms of means and standard deviations in Fig. 3 to show some detailed results on ten games. We choose those games both by alphabets and by representativeness. The first six games are selected alphabetically, and the remaining four are well-known classic Atari games. As the original paper of bootstrapped DQN provided full maximal evaluation scores on 49 Atari games, we present comparable full results of maximal evaluation scores and include them in Table II. To provide more advanced analysis over aggregated data, the measurement of performance profiles (score distributions) described in [1] with overall human normalized scores is shown Fig. 2.

To concisely represent contents in figures and tables, we use "*Boot-DQN*" as an abbreviation for the bootstrapped DQN algorithm in the original paper, "*Boot-DQN**" for our bootstrapped DQN implementation without using noise, and "*Boot-DQN+NP*" for bootstrapped DQN with priors replaced by noise. The only difference between "*Boot-DQN+NP*" and "*Boot-DQN**" is whether they use the noise (our algorithm) or not (the default algorithm).

In Fig. 3, the mean evaluation scores and the 95% confidence intervals of "*Boot-DQN+NP*" and "*Boot-DQN**" across different games are being compared. The 95% confidence intervals are obtained from five evaluation episodes for each step of a single training run. Oscillations of evaluation scores in both algorithms can be attributed to the different game initializations due to random seeds in the evaluation phase, and the stochasticity we introduce through no-op frames.

It is clear that our algorithm achieves not only higher final mean scores but also high mean evaluation scores during most of the training period than the default algorithm across the majority of games. In fact, the only game that "*Boot-DQN**" shows significantly better performance than "*Boot-DQN+NP*" is Amidar. "*Boot-DQN+NP*" beats "*Boot-DQN**" in almost all other games in terms of mean scores and confidence intervals.

At the beginning of each game, the differences of evaluation scores between these two algorithms are usually not significant. The default algorithm without using noise can even obtain better scores among some of the games, which can be attributed to the dithering effects of setting a lower boundary for $scales$, as described in Section II.

Occasionally, the default algorithm plateaus after approximately 1 million steps of training and its performances cannot improve meaningfully till the end of training. In those games, our algorithm with noise is less likely to find itself in a similar plateau, and even if it gets stuck, it is then able to demonstrate an entirely new behavior—breaking the plateau and achieving far higher scores at the end of training. Notably, our algorithm seems to be able to continue to keep increasing scores after 200 million frames in more games, given its better ability of escaping the plateau, which can be another promising feature.

In most cases, this "plateau" is not stationary and could accompany heavy drops in scores later in the training, due to the varied game difficulties along the episode caused by versatile game mechanisms. Both algorithms suffer from that and they may or may not recover from those downhills. Nevertheless, it is important to notice that "*Boot-DQN+NP*" vastly obtains higher mean evaluation scores with confidence intervals than "*Boot-DQN**" in most cases where this happens. Moreover, "*Boot-DQN+NP*" often bests "*Boot-DQN**" in final scores at the end of training, provided they are able to recover from the downhill.

Meanwhile, there are some games in which there is no obvious plateau at all. There seems to be constant improvements for both algorithms throughout the whole training process. Even so, "*Boot-DQN+NP*" shows faster convergence and demonstrates better performance than "*Boot-DQN**." This entails that noise can facilitate the environment exploration and finds improved

TABLE II
FULL RESULTS BY MAXIMAL EVALUATION SCORES

Game	Boot-DQN	Boot-DQN*	(Normalized)	Boot-DQN+NP	(Normalized)	DDQN	Nature
Alien	2436.6	7700	(1.083)	**12330**	(1.754)	4007.7	3069
Amidar	1272.5	873	(0.506)	728	(0.421)	**2138.3**	739.5
Assault	8047.1	8044	(15.053)	**8223**	(15.476)	6997.9	3359
Asterix	19713.2	40900	(4.906)	**46500**	(5.582)	17366.4	6012
Asteroids	1032	3320	(0.054)	**8960**	(0.177)	1984.4	1629
Atlantis	994500	1000700	(61.061)	**1005900**	(61.382)	767850	85641
Bank Heist	1208	**1300**	(1.74)	**1300**	(1.74)	1109	429.7
Battle Zone	38666.7	**61000**	(1.684)	**61000**	(1.684)	34620.7	26300
Beam Rider	23429.8	28850	(1.72)	**34936**	(2.087)	16650.7	6846
Bowling	60.2	30	(0.05)	**78**	(0.4)	77.9	42.4
Boxing	93.2	**100**	(8.325)	**100**	(8.325)	90.2	71.8
Breakout	**855**	834	(28.899)	822	(28.483)	437	401.2
Centipede	4553.5	**13980**	(1.2)	12383	(1.037)	4855.4	8309
Chopper Command	4100	3200	(0.363)	2400	(0.242)	5019	**6687**
Crazy Climber	137925.9	195000	(7.354)	**208000**	(7.873)	137244.4	114103
Demon Attack	82610	28475	(15.571)	35255	(19.3)	**98450**	9711
Double Dunk	3	**4**	(10.273)	2	(9.364)	-1.8	-18.1
Enduro	1591	2286	(2.657)	**2364**	(2.747)	1496.7	301.8
Fishing Derby	26	**41**	(2.504)	39	(2.466)	19.8	-0.8
Freeway	33.9	26	(0.878)	**34**	(1.149)	33.4	30.3
Frostbite	2181.4	**4690**	(1.083)	2430	(0.554)	2766.8	328.3
Gopher	17438.4	**118380**	(54.816)	118340	(54.797)	13815.9	8520
Gravitar	286.1	**2000**	(0.575)	1050	(0.276)	708.6	306.7
Hero	**21021.3**	13415	(0.416)	13375	(0.414)	20974.2	19950
Ice Hockey	-1.3	2	(1.091)	**3**	(1.174)	-1.7	-1.6
Jamesbond	1663.5	**7600**	(27.652)	7000	(25.46)	1120.2	576.7
Kangaroo	14862.5	14500	(4.843)	**16600**	(5.547)	14717.6	6740
Krull	8627.9	**11430**	(9.21)	10764	(8.587)	9690.9	3805
Kung Fu Master	36733.3	52800	(2.337)	**56900**	(2.52)	36365.7	23270
Montezuma Revenge	**100**	0	(0)	0	(0)	0	0
Ms Pacman	2983.3	5620	(0.8)	**5970**	(0.852)	3424.6	2311
Name This Game	11501.1	13950	(2.025)	**15180**	(2.239)	11744.4	7257
Pong	20.9	**21**	(1.181)	**21**	(1.181)	20.9	18.9
Private Eye	1812.5	**15100**	(0.217)	4000	(0.057)	158.4	1788
Qbert	15092.7	26375	(1.972)	**27600**	(2.064)	15209.7	10596
Riverraid	12845	15760	(0.914)	**17320**	(1.013)	14555.1	8316
Road Runner	51500	70500	(8.998)	**82100**	(10.479)	49518.4	18257
Robotank	66.6	54	(5.34)	52	(5.134)	**70.6**	51.6
Seaquest	9083.1	28830	(0.555)	**30800**	(0.732)	19183.9	5286
Space Invaders	2893	1950	(1.185)	2100	(1.284)	**4715.8**	1976
Star Gunner	55725	55300	(5.7)	60900	(6.284)	**66091.2**	57997
Tennis	0	1	(1.6)	1	(1.6)	**11.8**	-2.5
Time Pilot	9079.4	**15700**	(7.303)	8600	(3.029)	10075.8	5947
Tutankham	214.8	391	(2.43)	**399**	(2.481)	268	186.7
Up N Down	26231	**70120**	(6.235)	56380	(5.004)	19743.5	8456
Venture	212.5	**1700**	(1.432)	0	(0)	239.7	380
Video Pinball	811610	**999004**	(56.543)	894533	(50.63)	685911	42684
Wizard of Wor	6804.7	18700	(4.325)	**24500**	(4.683)	7655.7	3393
Zaxxon	11491.7	**17200**	(1.878)	14100	(1.539)	12947.6	4977

trajectories faster. This is especially true at the late training stage when ϵ-greedy is almost not at play, despite the fact that noise is not designed to replace ϵ-greedy at the first place.

In Fig. 4, we show the maximal Q-values predicted by the policy network alongside the period of training. It can be seen that adding noise increases the maximal Q-value predictions but not to an extent where it crashes the model by overestimation. In fact, the Q-values are still underestimated in general, and the influences of adding noise are more salient in games where severe underestimation occurs. In the game of Asteroids, where "*Boot-DQN*$*$" unstable behaviors and inflated Q-values, the maximal Q-values of "*Boot-DQN+NP*" surprisingly remain stable and still maintain smaller at the end. In Tennis, the maximal Q-values of "*Boot-DQN*$*$" overestimates to an absurdly large value, whereas those of "*Boot-DQN+NP*" still remain at reasonable values. Results from both games demonstrate that adding noise enhances the stability of our algorithm and "*Boot-DQN+NP*" even possesses the ability to achieve better performance than "*Boot-DQN*$*$" due to this improvement. This improvement is visible up to 200 M frames in tennis but does not remain till the end in Asteroids. Although it is not always the case that this improvement can be sustainable throughout the whole training process, it still results in higher maximal evaluation scores. In our view, adding bias to the model in this way serves as an additional regularizer and can be an excellent example of beneficial variance-bias tradeoff.

In Table II, the results for "*Boot-DQN*" and "*DDQN*" are taken from [22]. "*DDQN*" is their improved implementation

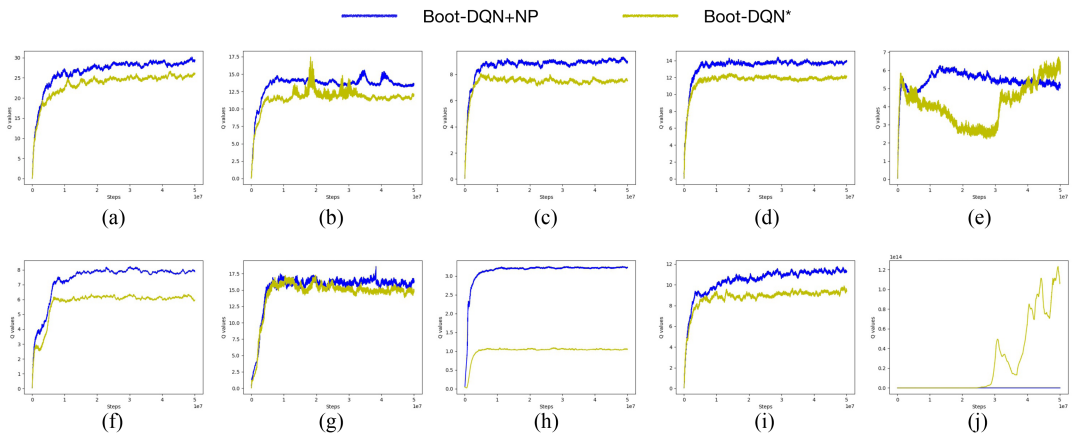


Fig. 4. Maximal Q-values corresponding to Fig. 3. Maximal Q-values are updated each time the network gets updated. Q-values are from the predictions of the policy network. Those values are used to update the parameter $scale$. (a) Q-values: Alien. (b) Q-values: Amidar. (c) Q-values: Assault. (d) Q-values: Asterix. (e) Q-values: Asteroids. (f) Q-values: Atlantis. (g) Q-values: Breakout. (h) Q-values: Freeway. (i) Q-values: Space Invaders. (j) Q-values: Tennis.

of the DQN with their own specified parameters. In contrast, *Nature* is the original DQN implementation from [19], which performs slightly worse than "*DDQN*." It might appear that "*Boot-DQN**" obtains better results than the original "*Boot-DQN*" by merely using different parameters in various games. In some other games, however, there are degraded performances for "*Boot-DQN**" or no significant differences between the two of them. In Freeway, the maximal evaluation score of "*Boot-DQN**" drops tremendously from 34 to around 26. Moreover, it is likely that the original "*Boot-DQN*" used averages of maximal scores, either by rolling averages or by multiple runs. Thus, it is important to direct our focus toward how using noise improves the performance of bootstrapped DQN (i.e., difference between "*Boot-DQN**" and "*Boot-DQN+NP*") but not toward parameter tunings and calibrations.

We highlight the algorithm which has achieved the best scores. Table II clearly shows that "*Boot-DQN+NP*" achieves better results compared to both "*Boot-DQN*" and "*Boot-DQN**" in terms of the maximal evaluation scores. The benefit of introducing noise alone boosts the performances of the bootstrapped DQN algorithm tremendously. In fact, the performance of scores might be even larger if by checking some of the intermediate results in Fig. 3. The maximal evaluation scores for our algorithm are higher than or equal to those of "*Boot-DQN**" in 30 out of 49 (61.2%) games.

Performance profiles give an aggregated score distribution on all tasks as a whole that allows qualitative comparisons. In Fig. 2, it is clear that "*Boot-DQN+NP*" has values similar to " *Boot-DQN**" at the beginning but significantly higher values approximately after $\tau = 2$. "*Boot-DQN+NP*" having a larger tail distribution entails an overall improved superhuman performance the algorithm is capable of achieving when $\tau > 2$.

V. Discussion

The results show that our algorithm outperforms both the original algorithm and the algorithm without using noise in most of the games. However, it is not the case that this performance enhancement can occur amongst all games. In ALE, there are a variety of different types of games, with distinct game mechanisms, such as goal and subgoal designs, reward assignments, agent and object controls. Those all imply that model-free RL methods without specific tuning or domain knowledge for each game can exhibit superhuman performances in a set of games but not in others. Nonetheless, it is of the interest of AI researchers to avoid using specific tuning or domain knowledge per game, and to let an algorithm learn to solve various tasks at the human or superhuman level. Naturally, our algorithm can perform better in many games but not in certain ones. What is crucial is that we have exactly demonstrated that *Boot-DQN+NP* yields improved performance than the default algorithm without using noise in the majority of the game results.

Our hypothesis indicates that such improvement on the performances of *Boot-DQN+NP* originates from its expanded search horizon. *Boot-DQN+NP* is capable of shifting the current spotlight from endlessly searching the attended trajectory to exploring other trajectories with seemingly lower rewards. However, there is no guarantee that the agent would direct its search efficiently toward the global optimum even if this optimum is within its expanded horizon. If the agent was able to return to a state with high rewards and promising discoveries first and then explore by some efficient strategies, its scores could potentially increase drastically. Thus, adding a policy-based Go-Explore mechanism is expected to further bolster our results [6].

In Fig. 3, both agents find themselves in plateaus without reaching the possible maximal scores in a variety of games. They can escape this plateau sometimes, but it often requires a large amount of training episodes. Ideally, one might wish to tune β a bit higher in order to reduce the amount of the period in which the agent gets stuck on the plateau. However, increasing the parameter β slightly in our experiment would not demonstrate more significant improvement on the capability of escaping plateaus, and increasing it too much would cause the

TABLE III
MEAN AND STD OF EVALUATION SCORES AFTER 200 M FRAMES

Game	Boot-DQN*	Std	Normalized	Boot-DQN+NP	Std	Normalized
Alien	2358.0	511.48	0.31	**3414.0**	158.19	0.46
Amidar	401.0	24.79	0.23	**545.6**	42.77	0.31
Assault	1207.6	270.02	1.9	**1663.4**	638.94	2.77
Asterix	19340.0	14808.73	2.31	**32820.0**	3263.37	3.93
Asteroids	**1030.0**	89.44	0.01	452.0	223.11	-0.01
Atlantis	**873180.0**	4752.43	53.18	859960.0	8920.0	52.36
Bank Heist	**566.0**	108.0	0.75	0.0	0.0	-0.02
Battle Zone	**31600.0**	3666.06	0.84	19800.0	2227.11	0.5
Beam Rider	11816.4	3616.03	0.69	**13762.0**	8137.31	0.81
Bowling	**30.0**	0.0	0.05	9.0	9.3	-0.1
Boxing	96.0	3.74	7.99	**98.4**	1.96	8.19
Breakout	**403.0**	7.97	13.93	398.6	7.61	13.78
Centipede	**2115.0**	565.45	0.0	2094.0	998.77	0.0
Chopper Command	880.0	97.98	0.01	**1080.0**	146.97	0.04
Crazy Climber	118340.0	4412.53	4.29	**146180.0**	7049.65	5.41
Demon Attack	6455.0	3324.02	3.47	**10748.0**	12470.36	5.83
Double Dunk	-5.6	4.8	5.91	**-1.6**	1.5	7.73
Enduro	1171.8	164.87	1.36	**1353.6**	35.89	1.57
Fishing Derby	-31.4	14.88	1.14	**-26.6**	7.94	1.23
Freeway	21.6	1.2	0.73	**33.2**	0.4	1.12
Frostbite	**2922.0**	693.26	0.67	744.0	78.38	0.16
Gopher	3880.0	1048.2	1.68	**44208.0**	32534.94	20.4
Gravitar	70.0	140.0	-0.03	**350.0**	303.32	0.06
Hero	3010.0	0.0	0.07	**7668.0**	110.12	0.22
Ice Hockey	-15.0	2.45	-0.31	**-4.6**	1.85	0.55
Jamesbond	**720.0**	156.84	2.52	470.0	40.0	1.61
Kangaroo	**11620.0**	943.19	3.88	10920.0	2584.88	3.64
Krull	7088.8	1695.02	5.14	**9295.2**	461.11	7.21
Kung Fu Master	2800.0	1395.71	0.11	**23640.0**	2081.92	1.04
Montezuma Revenge	**0.0**	0.0	0.0	**0.0**	0.0	0.0
Ms Pacman	1786.0	232.0	0.22	**3682.0**	921.88	0.51
Name This Game	**10072.0**	1985.32	1.35	9516.0	2868.68	1.25
Pong	**21.0**	0.0	1.18	20.6	0.8	1.17
Private Eye	0.0	0.0	-0.0	**100.0**	0.0	0.0
Qbert	**22215.0**	1898.0	1.66	21475.0	2907.71	1.6
Riverraid	10288.0	2686.7	0.57	**12902.0**	1582.46	0.73
Road Runner	43120.0	13120.27	5.5	**60240.0**	7593.05	7.69
Robotank	21.6	7.39	2.0	**31.0**	8.65	2.97
Seaquest	**17124.0**	1262.53	0.41	5852.0	1946.34	0.14
Space Invaders	704.0	409.5	0.37	**844.0**	537.52	0.46
Star Gunner	33560.0	6499.42	3.43	**36640.0**	7793.23	3.75
Tennis	-14.6	7.31	0.59	**-1.2**	0.4	1.46
Time Pilot	**5200.0**	1255.39	0.98	2860.0	1763.63	-0.43
Tutankham	0.0	0.0	-0.07	**261.6**	61.73	1.6
Up N Down	4432.0	153.41	0.35	**4736.0**	161.44	0.38
Venture	**420.0**	146.97	0.35	0.0	0.0	0.0
Video Pinball	223341.4	104619.78	12.64	**298420.6**	61065.4	16.89
Wizard Of Wor	460.0	80.0	-0.02	**1560.0**	80.0	0.24
Zaxxon	6140.0	1934.53	0.67	**8880.0**	1203.99	0.97

model to crash due to overestimation. Besides, a complex game often demands a sufficient amount of explorations in order for the agent to learn new behaviors in the game.

We distinguish our algorithm from the famous NoisyNet [8] in several ways. First, NoisyNet was proposed to replace the conventional exploration schemes such as ϵ-greedy and entropy reward, whereas our algorithm does not replace those exploration strategies. We still use ϵ-greedy to guide our explorations phase. Moreover, NoisyNet utilizes a parametric function to add noise into the weight and bias of their NN. It directly modifies the network architecture and introduces additional trainable parameters. On the other hand, our algorithm aims at improving the diversity of bootstrapped DQN. It combines the noise with target values, which does not require any modifications to the network architecture, and hence not introduce any additional computational overhead on the network level.

In short, NoisyNet works on a more general RL topic and tries to find replacements for exploration schemes at the costs of more algorithmic complexity, whereas our method improves on a specific algorithm, bootstrapped DQN, with adding minimal computations.

VI. CONCLUSION

In this article, we try to mitigate a problem common in bootstrapped DQN, where the diversity degrades after a tremendous amount of training. A fixed prior network has been proposed to mediate this problem. Nonetheless, the integrity of diversity still cannot be maintained in the late training phase with this modification despite being eased slightly. Our algorithm introduces randomness into the diversity constantly of bootstrapped DQN through noise. Priors here are being replaced by noise, with minimal extra computational overhead.

Our results demonstrate that our adjustment of the algorithm contributes to the improvements of evaluation scores in most of the 49 games we have trained and tested. Thus, our hypothesis

has been substantiated and the benefits of utilizing noise are significant.

Meanwhile, it has been pointed out that sticky actions can reliably introduce more stochasticity in Atari games [17]. It is beneficial to implement sticky actions in further experiments.

Besides using noise, there are other interesting concepts such as applying self-supervised learning to RL, which aspires to building world models of background knowledge. It could potentially help the agent plan and explore the environment [11], [29]. Bootstrapped DQN utilizes the random initialization of network priors to perform random exploration on rarely trained states, but it certainly could benefit from a world model. The combination of using noise and a world model could be an interesting future work.

Decision transformer has also shown the potential of solving RL tasks efficiently offline [2], which can be viewed as a successful combination of self-supervised learning and RL. Transformers, in a broad sense, share some similar concepts with bootstrapped DQN because they both use multiple network heads and also masking/gating mechanisms to direct the diversity or the attention of their networks. Although bootstrapped DQN found out that masking in practice cannot improve the performances but decreases the training speed, it might be interesting to adjust the mechanisms in accordance with those of transformers and to explore the undiscovered effects of masking more in the future.

Self-supervised learning and, more specifically, transformers, can be a potential future research direction to further improve the performance of bootstrapped DQN with noise. On the other hand, concepts from bootstrapped DQN with noise might potentially benefit the research of self-supervised learning as well.

APPENDIX

Evaluation Results After 200 M Frames

Mean scores, standard deviations, and human normalized scores after 200 M frames of training are reported in Table III, which is a supplement of Fig. 3. In general, the mean scores are considerably lower than the maximal evaluation scores demonstrated in Table II, with large standard deviations, accounted both by the small number (5) of evaluation runs and by the stochasticity. *Boot-DQN+NP* scores equal or higher than *Boot-DQN** in 33 out of 49 (67.3%) games, which is a bit higher than that in Table II.

ACKNOWLEDGMENT

The authors would like to acknowledge the help received from the Department for Research Computing at USIT, the University of Oslo IT-Department.

This work was performed on the [ML node] resource, owned by the University of Oslo, and operated by the Department for Research Computing at USIT, the University of Oslo IT-Department.[2]

[2][Online]. Available: http://www.hpc.uio.no/.

REFERENCES

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 29304–29320.

[2] L. Chen et al., "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 15084–15097.

[3] Y. Chen, L. Schomaker, and M. A. Wiering, "An investigation into the effect of the learning rate on overestimation bias of connectionist Q-learning," in *Proc. 13th Int. Conf. Agents Artif. Intell.*, 2021, pp. 107–118, doi: 10.5220/0010227301070118.

[4] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.

[5] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: A new approach for hard-exploration problems," 2019, *arXiv:1901.10995*.

[6] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, 2021.

[7] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *Proc. Int. Conf. Learn. Representations*, 2019.

[8] M. Fortunato et al., "Noisy networks for exploration," in *Proc. Int. Conf. Learn. Representations*, 2018.

[9] A. Guez, D. Silver, and P. Dayan, "Efficient Bayes-adaptive reinforcement learning using sample-based search," *Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.

[10] H. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, vol. 23. Red Hook, NY, USA: Curran Associates, 2010, pp. 2613–2621.

[11] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 15663–15674.

[12] T. Hester et al., "Deep Q-learning from demonstrations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3223–3230.

[13] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, vol. 29. Red Hook, NY, USA: Curran Associates, 2016, pp. 4565–4573.

[14] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, no. 4, pp. 1563–1600, 2010.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

[16] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds Mach.*, vol. 17, no. 4, pp. 391–444, 2007.

[17] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *J. Artif. Intell. Res.*, vol. 61, pp. 523–562, 2018.

[18] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning," *Proc. NIPS Deep Learn. Workshop*, 2013.

[19] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] A. Y. Ng et al., "Algorithms for inverse reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 1, 2000, pp. 663–670.

[21] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," *Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[22] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 4026–4034.

[23] I. Osband and B. Van Roy, "Bootstrapped Thompson sampling and deep exploration," 2015, *arXiv:1507.00300*.

[24] I. Osband et al., "Deep exploration via randomized value functions," *J. Mach. Learn. Res.*, vol. 20, no. 124, pp. 1–62, 2019.

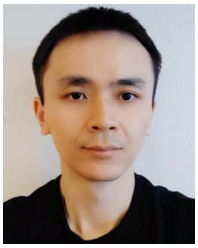
[25] I. Osband, B. Van Roy, and Z. Wen, "Generalization and exploration via randomized value functions," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2377–2386.

[26] M. Plappert et al., "Parameter space noise for exploration," in *Proc. Int. Conf. Learn. Representations*, 2018.

[27] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on Thompson sampling," in *Foundations and Trends in Machine Learning*. MA, USA: Now Publishers, Inc., vol. 11, 2018, pp. 1–96.

[28] T. Schaul, J. Togelius, and J. Schmidhuber, "Measuring intelligence through games," 2011, *arXiv:1109.1314*.

[29] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 8583–8592.
[30] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.
[31] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proc. 4th Connectionist Models Summer Sch.*, 1993, pp. 255–263.
[32] A. Touati, H. Satija, J. Romoff, J. Pineau, and P. Vincent, "Randomized value functions via multiplicative normalizing flows," in *Proc. Uncertainty Artif. Intell.*, 2020, pp. 422–432.
[33] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[34] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1995–2003.
[35] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.



Li Meng received the B.Sc. and M.Sc. degrees in artificial intelligence from the University of Groningen, Groningen, Netherlands, in 2017 and 2020, respectively. He is currently working toward the Ph.D. degree, working on reinforcement learning and computer vision, with the Department of Technology Systems, University of Oslo, Oslo, Norway.

Morten Goodwin received the B.Sc. degree in computer science and M.Sc. degree in information and communication technology from the University of Agder, Grimstad, Norway, in 2003 and 2005, respectively, and the Ph.D. degree in computer science from the Department of Computer Science, Aalborg University, Aalborg, Denmark, in 2011, on applying machine learning algorithms on eGovernment indicators which are difficult to measure automatically.

He is currently a Professor with the Department of ICT, University of Agder, Deputy Director with the Centre for Artificial Intelligence Research, an adjunct Professor with OsloMet, CTO AIVEO, a public speaker, and an author. His main research interests include deep learning and reinforcement learning. He does fundamental and applied AI research within agriculture, aquaculture, cultural production, education, health, industry optimization, natural language processing, and recommendation engines.

Anis Yazidi (Senior Member, IEEE) received the M.Sc. degree in information and communication technology and Ph.D. degree in artificial intelligence from the University of Agder, Grimstad, Norway, in 2008 and 2012, respectively.

He is currently the Deputy Head with OsloMet AI Lab, Oslo, Norway, and the leader for the research group on Applied Artificial Intelligence (AI2), OsloMet. He is a Full Professor in machine learning with OsloMet. He is also a Senior Researcher with Oslo University Hospital (OuS), Oslo, and a Research Professor in data science with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway. He has more than 190 publications, including more than 80 journal articles in prestigious venues and two book chapters. He is leading the master's program in data science at OsloMet. He is the Codirector of the Excellence Academic Environment NordSTAR on Trustworthy and Sustainbe AI at OsloMet.

Dr. Yazidi was selected as the promising Researcher of the year, TKD Faculty OsloMet in 2015. In 2019, he won the prize for top 50 in Norway Most Productive Researcher for all disciplines for the years 2015–2018. He also won the Best Paper Award in SMARTGIFG 2017, in ACM RACS 2017, and in CSE 2014, and the Best Paper Award Runner in SMC 2016. He is an Associate Editor for the Springer Journal on *Pattern Analysis and Applications*, for *Frontiers in Artificial Intelligence*, and for *Frontiers in Computational Physiology and Medicine*. He is currently PI in the Horizon 2020 AI-Mind project and he is also PI from the Norwegian side in different international and national projects.

Paal E. Engelstad received the bachelor's degree in physics and mathematics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 1993, the master's degree in physics from NTNU/Kyoto University, Kyoto, Japan, in 1994, and the Ph.D. degree in computer science from the University of Oslo, Oslo, Norway, in 2005.

He is currently a Full Professor in computer science with the Department of Technology Systems, University of Oslo, where he is leading the research section for autonomous systems and sensor technologies. His research interests include distributed systems, security, autonomous systems, and machine learning. He also holds a 20% Professor position at Oslo Metropolitan University, Oslo.