

Chess Assignment

Introduction to Reinforcement Learning (Spring 2022). Source code available at: https://github.com/TwoDigitsOneNumber/IntroRL_ChessAssignment

van den Bergh Laurin

Institute of Informatics

University of Zurich

Zurich, Switzerland

laurin.vandenbergh@uzh.ch — Matriculation number: 16-744-401

Abstract—We explored the use of three deep reinforcement learning methods for training agents to play a simplified version of chess. All algorithms, SARSA, Q-Learning and DQN were able to learn successful strategies. DQN was able to overcome the shortcomings of Q-Learning and provides an off-policy method with performance comparable to the on-policy method SARSA.

Index Terms—deep reinforcement learning, chess, temporal-difference methods

I. INTRODUCTION

In this assignment, we explore three different deep reinforcement learning algorithms to learn how to play a simplified version of chess, which can be thought of as a special instance of an endgame. These three algorithms are SARSA, Q-Learning and DQN¹. We will first provide a general look over our methodology (see Section II) and later discuss the result obtained in our experiments (see Section III).

The focus of this report lies on comparing these three algorithms in theory and in practise on the chess endgame environment. We further explored the impact of the hyper-parameters β and γ , which represent the speed of the decaying trend for the learning rate and the discount factor respectively.

Throughout the report we indicate in footnotes which task a particular section is referring to in terms of answering the task. We do this as the solutions to certain tasks are spread throughout multiple sections, e.g. task 3 is answered in Section II and Section III. Even though this assignment was not solved in a group, we decided to also answer some of the “group only” and we stick to the numbering of the assignment in order to avoid confusion.

II. METHODS

A. Environment

This version of chess takes place on a 4 by 4 board and can be thought of as a specific version of an endgame where the agent has a king and a queen, and the opponent has only a king. Since this game can only end in a win for the agent or in a draw, it is the agent’s goal to learn how to win the game and avoid draws. For all experiments considered, the agent will be given a reward of 1 for winning, 0 for drawing, and 0 for all intermediate steps.

¹SARSA serves as answer to task 3 and DQN serves as answer to task 5.

This chess setting, and chess in general, fulfills the Markov property and therefore justifies the use of the temporal difference methods used in this assignment.

B. SARSA and Q-Learning²

1) *Temporal-Difference Algorithms*: SARSA and Q-Learning are two very related model-free types of temporal-difference (TD) algorithms for learning expected rewards, also known as Q-values, when rewards are not immediate and possibly sparse. The learning takes place via interaction with an environment through trial and error. These Q-values are in general represented by an action-value function Q and, for finitely many state-action pairs (s, a) , can be considered as a Q-table where each state-action pair, (s, a) , maps to a single Q-value, thus providing an estimate of the quality of any given state-action pair (s, a) . In this assignment however we use neural networks to approximate the action-value function, which outputs the Q-values for all possible actions for any given state. This helps to avoid computing large Q-tables. All algorithms explored in this assignment, including DQN, require the environment to fulfill the Markov property.

2) *On-policy vs. Off-policy*: SARSA and Q-Learning address the temporal-credit-assignment problem [1], that is, trying to attribute future rewards to previous actions. These future rewards get discounted with the hyper-parameter γ (see Section III-C). Both algorithms repeatedly choose and take actions in the environment according to some policy π , e.g. an ϵ -greedy policy.

However, this is where they differ. SARSA is an on-policy algorithm, which means that consecutive actions are chosen according to the same policy π , even during the update step of the Q-values, which leads to the update rule:

$$Q_{\pi}(s, a) \leftarrow Q_{\pi}(s, a) + \eta(r + \gamma Q_{\pi}(s_{t+1}, a') - Q_{\pi}(s, a))$$

for some future action a' chosen according to policy π .

Q-learning, on the other hand, is an off-policy algorithm, which means that it takes its actions a according to its policy π , but during the update steps it assumes a greedy policy, i.e. optimal play, for future actions a' . Q-Learning has the update rule:

$$Q_{\pi}(s, a) \leftarrow Q_{\pi}(s, a) + \eta(r + \gamma \max_{a'} Q_{\pi}(s_{t+1}, a') - Q_{\pi}(s, a)).$$

²Answer to task 1.

3) *Advantages and Disadvantages*: This leads to one of Q-Learning’s major advantages: Because of Bellman’s optimality equation, Q-Learning is guaranteed to learn the values for the optimal policy, i.e. $Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$, regardless of the policy used to train it, and in a greedy setting will take the optimal actions, at least if it was trained sufficiently. However, this can in certain cases mean that the online performance of Q-Learning will be worse than the one from SARSA, as Sutton et al. [2] demonstrate with their “gridworld” example “Cliff Walking”. Our chess game is a similar situation, because a win and a draw can be very close, thus during exploration Q-Learning can accidentally create a draw because it is going for the optimum when exploiting. Q-Learning is however relatively unstable and the parameters can even diverge when it is combined with non-linear function approximators [3], making the guarantee to learn the optimal policy irrelevant.

SARSA will learn to take a safer path, because it keeps its policy in mind when updating the Q-values, i.e. it keeps in mind that it will explore in future actions. This has the advantage that SARSA in general tends to explore more than Q-Learning.

C. Experience Replay³

Experience replay is a technique proposed by Lin [4] to speed up the training process for reinforcement learning algorithms by reusing past experiences for future training. This is analogous to the human ability to remember past experiences and learn from them even after the fact. The past experiences are stored in a replay memory of fixed size at each time step t as a tuple $e_t = (s_t, a_t, r_t, s_{t+1})$. This essentially allows us to transform the learning process from online learning to mini-batch learning, where a batch of experiences e_j is randomly sampled for each update step. Experience replay can only be used in combination with off-policy algorithms, because otherwise the current parameters determine the next sample and create unwanted feedback loops [3], [5].

Experience replay provides many benefits over online Q-Learning, especially when neural networks are used to approximate the action-value function. First, it enables the agent to learn from past experiences more than once, leading to increased data efficiency and faster convergence [4], [5]. Second, since for each update step past experiences are sampled randomly, the correlations between the individual actions are reduced, which then reduces the variance of the updates [5]. This leads to the experience samples e_j being closer to i.i.d. and thus guaranteeing better convergence when using optimization algorithms such as stochastic gradient descent as most convergence proofs assume i.i.d. data.

D. Deep Q-Networks (DQN)⁴

A first version of the DQN algorithm was proposed by Mnih et al. [5] and combined experience replay with Q-learning, where a neural network was used as a non-linear function approximator for the action-value function. Mnih et al. [3] later

improved upon the method and presented the DQN algorithm, as it is known today, where they address the problem of the Q-values $Q_{\pi}(s, a)$ being correlated to the target values $y = r + \gamma \max_{a'} Q_{\pi}(s', a')$ because they are generated using the same neural network. In the DQN algorithm they separated the Q-network from the target network and only update the target network every C steps, which helps to break this correlation and combat diverging network parameters.

Since DQN uses experience replay, we essentially transform the reinforcement learning task to a supervised learning task. Therefore a suitable loss function for the neural network is needed. Mnih et al. [3] used a squared loss of the temporal-difference error, also known as delta: $\delta = y - Q_{\pi}(s, a)$.

E. Experiments

In order to address all tasks, we divided the tasks into several independent experiments. First, we conducted seeded runs⁵ for all three algorithms using seed 21 for reproducibility, which was chosen a-priori. These seeded runs serve as examples to compare the algorithm’s online performance qualitatively. The seeds are used such that the weights of all neural networks are instantiated identically for all algorithms and they subsequently serve as seeds for any random number used during training. This makes sure that all agents start with the same initial conditions and that the results are reproducible (see Section V-1). All algorithms were run for 100000 episodes using identical model architecture and hyper-parameters (see Section II-F).

Since the seeded runs are heavily influenced by the choice of the seed, we could end up with anything between a very lucky and well performing seed, or with a very unlucky one. Also the interpretation of the seeded runs is more difficult as we just have one run for each algorithm. Therefore, we decided to perform a simulation study and complete 30 non-seeded runs for each algorithm in order to get a better idea of how the algorithms perform on average. For computational reasons we limited these runs to 40000 episodes as we realized with test runs that by then most of the training progress has already taken place.

To analyze the impact of the hyper-parameters β and γ ⁶ we trained 49 agents with different combinations for β and γ but keeping all other hyper-parameters and model architecture identical. We chose SARSA for this experiment as we found it to have very low variance between its unseeded runs, which makes it an ideal candidate for comparing individual runs. These runs are seeded identically to the seeded runs mentioned above.

F. Implementation and Hyper-parameters

We implemented all algorithms from scratch according to Sutton et al. [2] (SARSA and Q-Learning⁷) and Mnih et al. [3] (DQN⁸). For the implementation see file `neural_net.py`

³Answer to “group only” task 2.

⁴Answer to task 5: Describing the used method.

⁵Answers to task 3 and 5.

⁶Answer to task 4.

⁷SARSA as answer to task 3 and Q-Learning as additional algorithm.

⁸Answer to task 5.

on GitHub or Listing 1. All algorithms use a neural network with 58 input neurons, 200 hidden neurons and 32 output neurons, not including the biases for the input and hidden layer. The neural network automatically adds a constant input for the bias and the hidden layer. The implementation treats the biases like any other weights and thus they are part of any matrix multiplication. We used a ReLU activation function for the hidden layer and no activation on the output layer. The weights were initialized using Glorot initialization [6], such that the weights are sampled from a normal distribution with mean 0 and variance $\frac{2}{n_{in}+n_{out}}$, where n_{in} and n_{out} denote to the number of input and output neurons of the respective layer. This helped preventing exploding gradients for the most part. [check again with results](#)

For all experiments we used the default hyperparameters provided in the `Assignment.ipynb` file unless otherwise noted (see Table I). For DQN we updated the weights of the target network after every $C = 10$ steps, as most games take fewer steps than that. We used a replay memory of size 100000 and a batch size of 32.

Parameter	Value
Nr. input neurons	58+1
Nr. hidden neurons	200+1
Nr. output neurons	32
Initial exploration probability ϵ_0	0.2
Learning rate η	0.035
Decay rate of ϵ , β	0.00005
Discount factor γ	0.85

TABLE I: Common hyper-parameters shared by all algorithms.

III. RESULTS

A. Seeded Runs⁹

The rewards and number of moves for the seeded runs are depicted in Figures 1 and 2 respectively. Since the curves are very noisy, we smoothed them using an exponential moving average (EMA) with a weight on the most recent observation of $\alpha = 0.001$.

As expected, the online performance of Q-Learning in terms of the rewards is generally lower than the rewards for SARSA but they converge slowly as ϵ decreases (Figure 1). Also in Figures 1 and 2 we can see that Q-Learning experiences instable learning behavior as both plots are a lot more noisy and at about 20000 and 90000 episodes the rewards decrease for some period. SARSA and DQN don't show this behavior.

Even though the number of steps is not punished, all agents still learn to reduce the number of steps over time, as they do not give rewards and their goal is to take actions that do. SARSA seems to do the best job at this, which perhaps is caused by its tendency to explore more and find better strategies. Q-Learning however seems to struggle to reduce the number of steps it takes.

⁹Answer to task 3 (SARSA and Q-Learning as additional algorithm) and 5 (DQN).

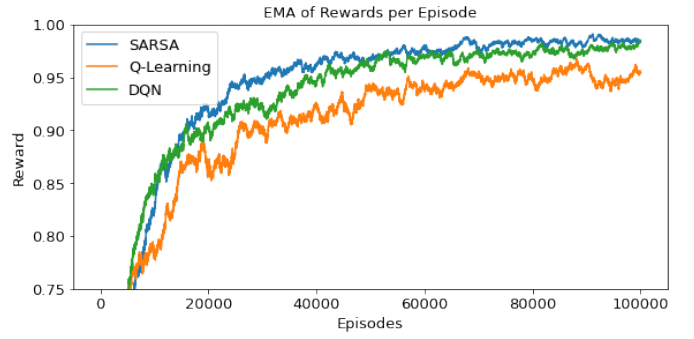


Fig. 1: Exponential moving average of the rewards achieved during training for 100000 episodes with identical hyper-parameters, weight initialization and model architecture.

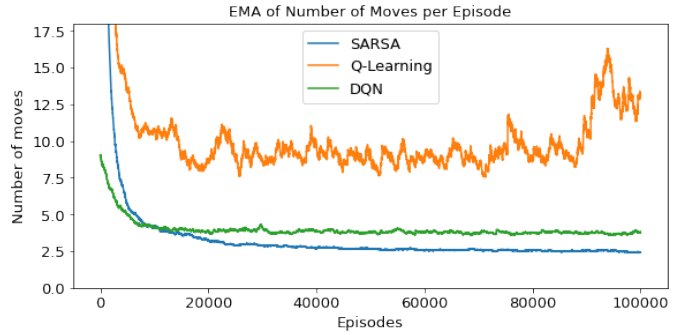


Fig. 2: Exponential moving average of the number of moves per episode achieved during training for 100000 episodes with identical hyper-parameters, weight initialization and model architecture.

As suggested by Mnih et al. [3], [5], DQN¹⁰ was able to overcome the downsides of Q-Learning which lead to an online performance which is comparable to that of SARSA in terms of reward and number of moves it achieved, and also in terms of the stability during training. It did however not learn to reduce the amounts of steps as much as SARSA.

B. Simulation Study (Non-seeded Runs)

We can confirm that the qualitative results from the seeded runs reasonably well represent the average case. The only notable exception being Q-Learning, for which most runs performed equally well to the seeded run, but some runs experienced huge increases in the number of steps, which influenced the average run dramatically, leading to an average of about 23 moves per episode after 40000 episodes.

We observed that DQN and SARSA show very comparable learning curves with DQN showing slightly faster convergence in the first 5000 episodes. SARSA showed the lowest variance in all runs and seems to be a very stable algorithm. Q-Learning on the other hand showed clear signs of divergence as for some runs the rewards consistently dropped while the number of moves consistently increased. This shows that the measures

¹⁰Answer to task 5 for comparing DQN to SARSA and Q-Learning.

taken by Mnih et al. [3] to combat the disadvantages of Q-Learning worked and increased the stability as well as the convergence speed. We were able to verify that the gradients of the Q-Learning agents were a lot less stable than the gradients of the other agents. However, using the Glorot initialization [6] helped prevent exploding gradients from occurring.

We also found out that, unsurprisingly, the effective training time is mainly dependent on the number of steps an algorithm takes per episode. This leads to Q-Learning having by far the longest training time, especially when the parameters diverge and the number of steps increase. DQN and SARSA have relatively short training times, with SARSA being the fastest.

We can conclude that the seeded runs in our initial experiment truthfully represent the average run and therefore some level of inference is justified.

C. Hyper-parameters¹¹

Figure 3(a) depicts the rewards and number of moves per episode as a function of β and γ . We can see that the reward increases monotonically as γ is increased, suggesting that a value of $\gamma \in [0.80, 1)$ should be chosen for almost all values of β . This intuitively makes sense, as we have very sparse rewards and want the agent to “backpropagate” this reward through its sequence of actions. The left plot of Figure 3 suggests that reducing γ to a value in $[0.5, 0.8]$ can teach the SARSA agent to not reduce the number of steps. Intuitively this makes sense, as the only reward will be “backpropagated” less to earlier states and thus the agent will move faster towards setting the opponent’s king checkmate.

We can not see a clear relationship between β and the rewards, apart from $\beta = 0$ being an inferior choice for all values of γ . In Figure 3(b) we can see that the number of steps taken by the agent decreases drastically when increasing γ from very low levels, but this effect seems larger for larger values of β . We can however see that there is a slight, but possibly insignificant, peak in the rewards around $\beta = 5 \cdot 10^{-3}$. In summary, for reasonably chosen values of γ the choice of β seems to not have much of an influence for training periods of around 40000 episodes.

IV. CONCLUSION

We are aware that the performance of the individual algorithms could be improved by tuning the hyper-parameters, however, this was not explicitly asked for and the focus on this assignment lies on the comparison of these algorithms from a theoretical and practical perspective.

For any deep reinforcement learning method the choice of suitable hyper-parameters for the task is crucial and can have large impacts on the training outcome. In our case, the default parameters provided to us performed very well so no need for much further consideration was necessary.

All three algorithms were able to learn to play the simplified version of chess to a very high degree even without hyper-parameter tuning. SARSA proved to be the most stable

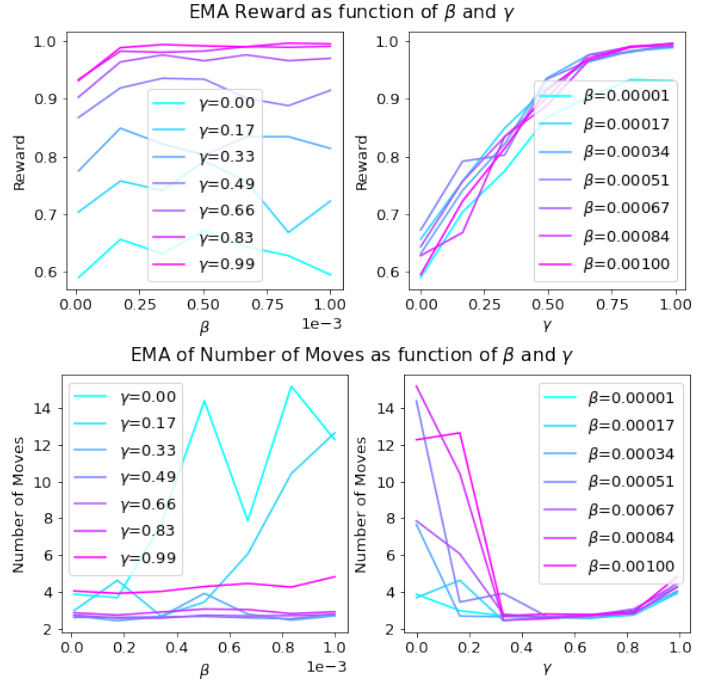


Fig. 3: Rewards and number of moves as functions of the speed of the decaying trend β and the discount factor γ after training a SARSA agent for 40000 episodes.

algorithm, which was confirmed to be the general case with 30 non-seeded runs. Q-Learning suffers from some instabilities when training, but DQN was able to overcome all of the problems of Q-Learning and provides an off-policy method that can learn with high stability, fast convergence and a low training time comparable to SARSA. Since DQN is an off-policy method, it comes with the added advantage that it will learn an optimal policy, similar to Q-Learning.

REFERENCES

- [1] R. S. Sutton, “Temporal credit assignment in reinforcement learning,” 1984.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: <http://incompleteideas.net/book/the-book-2nd.html>
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: <https://doi.org/10.1038/nature14236>
- [4] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Mach. Learn.*, vol. 8, no. 3–4, p. 293–321, may 1992. [Online]. Available: <https://doi.org/10.1007/BF00992699>
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [6] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics*, 2010.

¹¹ Answer to task 4.

V. APPENDIX

1) *Reproducibility:* In order to reproduce the results presented in this report we provide the code on GitHub in the repository https://github.com/TwoDigitsOneNumber/IntroRL_ChessAssignment. Along this we provide a conda environment file `environment.yaml` which can be used to recreate the exact environment we used. We recommend to run the file `Assignment_Train_Algorithms.ipynb` before the files `Assignment_Compare_Algorithms.ipynb` and `Assignment_Hyperparameter_Influence.ipynb` as the latter use files generated by the former. However, even on fast hardware running the former file takes between 5-6 hours, so we provide all necessary intermediate outputs in the repository as well.

```

1 # import libraries
2 from types import MethodDescriptorType
3 import numpy as np
4 from tqdm.notebook import tqdm
5 import os
6 import json
7 import time
8 import random
9 from collections import namedtuple, deque
10
11 # import from files
12 from Chess_env import *
13
14
15 # ===== Epsilon-greedy Policy =====
16
17 def EpsilonGreedy_Policy(Qvalues, allowed_a, epsilon):
18     """
19     returns: tuple
20         an action in form of a one-hot encoded
21         vector with the same shape as
22         Qvalues.
23         an action as decimal integer (0-based)
24
25     Assumes only a single state, i.e. online
26     learning and NOT (mini-)batch learning.
27     """
28     # get the Qvalues and the indices (relative of
29     # all Qvalues) for the allowed actions
30     allowed_a_ind = np.where(allowed_a==1)[0]
31     Qvalues_allowed = Qvalues[allowed_a_ind]
32
33     # ----- epsilon greedy -----
34
35     # draw a random number and compare it to epsilon
36     rand_value = np.random.uniform(0, 1, 1)
37
38     if rand_value < epsilon: # if the random number
39         is smaller than epsilon, draw a random
40         action
41         action_taken_ind_of_allwed_only = np.random.
42             randint(0, len(allowed_a_ind))
43     else: # greedy action
44         action_taken_ind_of_allwed_only = np.argmax(
45             Qvalues_allowed)
46
47     # get index of the action that was chosen (
48     # relative to all actions, not only allowed)
49     ind_of_action_taken = allowed_a_ind[
50         action_taken_ind_of_allwed_only]
```

```

# ----- create usable output -----
# get the shape of the Qvalues
N_a, N_samples = np.shape(Qvalues) # N_samples
must be 1
# initialize all actions of binary mask to 0
A_binary_mask = np.zeros((N_a, N_samples))
# set the action that was chosen to 1
A_binary_mask[ind_of_action_taken, :] = 1
return A_binary_mask, ind_of_action_taken

# ===== activation functions and it's derivatives
=====
# relu and its derivative
def relu(x):
    return np.maximum(0, x)
def heaviside(x):
    return np.heaviside(x, 0)
# sigmoid and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def gradient_sigmoid(x):
    return sigmoid(x) * (1 - sigmoid(x))
# tanh and its derivative
def tanh(x):
    return np.tanh(x)
def gradient_tanh(x):
    return 1 - np.tanh(x)**2
# identity and its derivative
def identity(x):
    return x
def const(x):
    return np.ones(x.shape)
def act_f_and_gradient(activation_function="relu"):
    if activation_function == "relu":
        return relu, heaviside
    elif activation_function == "sigmoid":
        return sigmoid, gradient_sigmoid
    elif activation_function == "tanh":
        return tanh, gradient_tanh
    else: # identity and constant 1
        return identity, const

# ===== Replay Memory for Experience Replay (with
DQN) =====
Transition = namedtuple('Transition', ("state", "
action", "reward", "next_state", "done"))
class ReplayMemory(object):
    def __init__(self, capacity):
        self.memory = deque(maxlen=capacity)
    def push(self, *args):
        self.memory.append(Transition(*args))
```

```

113 def sample(self, batch_size):
114     # if less data than batch size, return all
115     data
116     if len(self) < batch_size:
117         batch_size = len(self)
118     return random.sample(self.memory, batch_size)
119
120 def __len__(self):
121     return len(self.memory)
122
123 # ===== Neural Network =====
124
125 class NeuralNetwork(object):
126
127     def __init__(self, N_in, N_h, N_a,
128                 activation_function_1="relu",
129                 activation_function_2=None, method="
130                 qlearning", seed=None, capacity=100_000, C
131                 =100):
132         """
133         activation functions: "relu", "sigmoid", "
134         tanh", None
135         methods: "qlearning", "sarsa", "dqn"
136         """
137         self.D = N_in # input dimension (without
138         bias)
139         self.K = N_h # nr hidden neurons (without
140         bias)
141         self.O = N_a # nr output neurons (letter
142         , not digit 0)
143
144         # store method and seed
145         self.method = method
146         self.seed = seed
147
148         if self.method == "dqn":
149             self.capacity = capacity
150             self.replay_memory = ReplayMemory(
151                 capacity)
152             self.C = C
153
154         # set activation function and gradient
155         function
156         self.act_f_1_name = activation_function_1
157         self.act_f_2_name = activation_function_2
158         self.act_f_1, self.grad_act_f_1 =
159         self.act_f_and_gradient(activation_function_1)
160         self.act_f_2, self.grad_act_f_2 =
161         self.act_f_and_gradient(activation_function_2)
162
163         # initialize the weights and biases and set
164         global seed
165         np.random.seed(self.seed)
166
167         # self.W1 = np.random.randn(self.K+1, self.D
168         +1)/np.sqrt(self.D+1) # standard normal
169         distribution, shape: (K+1, D+1)
170         # glorot/xavier normal initialization
171         # self.W1 = np.random.randn(self.K+1, self.D
172         +1)*np.sqrt(2/ (self.D+1 + self.K+1)) #
173         standard normal distribution, shape: (K+
174         1, D+1)
175         self.W1 = np.random.standard_normal((self.K
176         +1, self.D+1))*np.sqrt(2/ (self.D+1 +
177         self.K+1)) # standard normal
178         distribution, shape: (K+1, D+1)
179         # self.W1 = np.random.randn(self.K+1, self.D
180         +1) # standard normal distribution,
181         shape: (K+1, D+1)
182
183         if self.method == "dqn":
184             self.W1_target = np.copy(self.W1)
185             self.W2_target = np.copy(self.W2)
186
187     def forward(self, x, target=False):
188         """
189         x has shape: (D+1, 1) (constant bias 1 must
190         be added beforehand)
191         target: if True, use the weights of the
192         target network
193
194         returns:
195             last logits (i.e. Qvalues) of shape (O,
196             1)
197         """
198         if target == True:
199             W1 = np.copy(self.W1_target)
200             W2 = np.copy(self.W2_target)
201         else:
202             W1 = np.copy(self.W1)
203             W2 = np.copy(self.W2)
204
205         # forward pass/propagation
206         a1 = W1 @ x
207         h1 = self.act_f_1(a1)
208         h1[0,:] = 1 # set first row (bias to second
209         layer) to 1 (this ignores the weights
210         for the k+1th hidden neuron, because
211         this should not exist; this allows to
212         only use matrix multiplication and
213         simplify the gradients as we only need 2
214         instead of 4)
215         a2 = W2 @ h1
216         h2 = self.act_f_2(a2)
217         return a1, h1, a2, h2
218
219     def backward(self, R, x, Qvalues, Q_prime, a1,
220                 h1, a2, gamma, future_reward,
221                 action_binary_mask):
222         """
223         backward for methods "qlearning" and "sarsa"
224
225         x has shape (D+1, 1) (constant bias 1 must
226         be added beforehand)
227         set future_reward=True for future reward
228         with gamma>0, False for immediate reward
229         .
230         Q_prime must be chosen according to the
231         method on x_prime (on- or off-policy)
232         """
233         # backward pass/backpropagation
234         # compute the gradient of the square loss
235         with respect to the parameters
236
237         # ===== compute TD error (aka delta) =====

```



```

211                                     272
212 # make reward of shape (O, 1)      273
213 R_rep = np.tile(R, (self.O, 1))    274
214 if future_reward: # future reward  275
215     delta = R_rep + gamma*Q_prime - Qvalues
216     # -> shape (O, 1)              276
217 else: # immediate reward            277
218     delta = R_rep - Qvalues # -> shape (O
219     1)                              278
220                                     279
221                                     280
222                                     281
223 # update only action that was taken, i.e. 282
224     all rows apart from the one
225     corresponding to the action taken (
226     action index) are 0              283
227                                     284
228 delta = delta*action_binary_mask      285
229                                     286
230                                     287
231 self.compute_gradients(delta, a1, h1, a2, 288
232 self.update_parameters(self.eta)      289
233                                     290
234 def backward_dqn(self, batch, gamma):  291
235     """                               292
236     backward for method "dqn"         293
237     """                               294
238                                     295
239 # ===== compute targets y and feature matrix 296
240 X =====                           297
241                                     298
242 # turn batch into individual tuples, numpy 299
243 arrays, or lists                    300
244 states = batch.state                301
245 rewards = np.array(list(batch.reward)) 302
246 actions = np.array(list(batch.action)) 303
247 next_states = list(batch.next_state)   304
248 dones = np.array(list(batch.done))     305
249                                     306
250 # compute targets y and feature matrix X    307
251 y = np.zeros((self.O, len(dones)))       308
252 for j in np.arange(len(dones)):          309
253     if dones[j]: # if done, set y_j = r_j 310
254         y[actions[j], j] = rewards[j]    311
255     else:                                   312
256         # compute Q_prime                313
257         Q_target = self.forward(next_states 314
258             j], target=True)[-1]          315
259         y[actions[j], j] = rewards[j] +   316
260             gamma*np.max(Q_target)        317
261                                     318
262 # convert states to feature matrix X       319
263 X = np.hstack((states))                 320
264                                     321
265 # ===== compute TD error (aka delta) ===== 322
266                                     323
267 al, h1, a2, Qvalues = self.forward(X)    324
268 delta = y - Qvalues # -> shape (O,
269     batch_size)                        325
270                                     326
271 self.compute_gradients(delta, a1, h1, a2, 327
272 self.update_parameters(self.eta)        328
273                                     329
274                                     330
275                                     331
276                                     332
277                                     333
278                                     334
279                                     335
280                                     336
281                                     337
282                                     338
283                                     339
284                                     340
285                                     341
286                                     342
287                                     343
288                                     344
289                                     345
290                                     346
291                                     347
292                                     348
293                                     349
294                                     350
295                                     351
296                                     352
297                                     353
298                                     354
299                                     355
300                                     356
301                                     357
302                                     358
303                                     359
304                                     360
305                                     361
306                                     362
307                                     363
308                                     364
309                                     365
310                                     366
311                                     367
312                                     368
313                                     369
314                                     370
315                                     371
316                                     372
317                                     373
318                                     374
319                                     375
320                                     376
321                                     377
322                                     378
323                                     379
324                                     380
325                                     381
326                                     382
327                                     383
328                                     384
329                                     385
330                                     386
331                                     387
332                                     388
333                                     389
334                                     390
335                                     391
336                                     392
337                                     393
338                                     394
339                                     395
340                                     396
341                                     397
342                                     398
343                                     399
344                                     400
345                                     401
346                                     402
347                                     403
348                                     404
349                                     405
350                                     406
351                                     407
352                                     408
353                                     409
354                                     410
355                                     411
356                                     412
357                                     413
358                                     414
359                                     415
360                                     416
361                                     417
362                                     418
363                                     419
364                                     420
365                                     421
366                                     422
367                                     423
368                                     424
369                                     425
370                                     426
371                                     427
372                                     428
373                                     429
374                                     430
375                                     431
376                                     432
377                                     433
378                                     434
379                                     435
380                                     436
381                                     437
382                                     438
383                                     439
384                                     440
385                                     441
386                                     442
387                                     443
388                                     444
389                                     445
390                                     446
391                                     447
392                                     448
393                                     449
394                                     450
395                                     451
396                                     452
397                                     453
398                                     454
399                                     455
400                                     456
401                                     457
402                                     458
403                                     459
404                                     460
405                                     461
406                                     462
407                                     463
408                                     464
409                                     465
410                                     466
411                                     467
412                                     468
413                                     469
414                                     470
415                                     471
416                                     472
417                                     473
418                                     474
419                                     475
420                                     476
421                                     477
422                                     478
423                                     479
424                                     480
425                                     481
426                                     482
427                                     483
428                                     484
429                                     485
430                                     486
431                                     487
432                                     488
433                                     489
434                                     490
435                                     491
436                                     492
437                                     493
438                                     494
439                                     495
440                                     496
441                                     497
442                                     498
443                                     499
444                                     500
445                                     501
446                                     502
447                                     503
448                                     504
449                                     505
450                                     506
451                                     507
452                                     508
453                                     509
454                                     510
455                                     511
456                                     512
457                                     513
458                                     514
459                                     515
460                                     516
461                                     517
462                                     518
463                                     519
464                                     520
465                                     521
466                                     522
467                                     523
468                                     524
469                                     525
470                                     526
471                                     527
472                                     528
473                                     529
474                                     530
475                                     531
476                                     532
477                                     533
478                                     534
479                                     535
480                                     536
481                                     537
482                                     538
483                                     539
484                                     540
485                                     541
486                                     542
487                                     543
488                                     544
489                                     545
490                                     546
491                                     547
492                                     548
493                                     549
494                                     550
495                                     551
496                                     552
497                                     553
498                                     554
499                                     555
500                                     556
501                                     557
502                                     558
503                                     559
504                                     560
505                                     561
506                                     562
507                                     563
508                                     564
509                                     565
510                                     566
511                                     567
512                                     568
513                                     569
514                                     570
515                                     571
516                                     572
517                                     573
518                                     574
519                                     575
520                                     576
521                                     577
522                                     578
523                                     579
524                                     580
525                                     581
526                                     582
527                                     583
528                                     584
529                                     585
530                                     586
531                                     587
532                                     588
533                                     589
534                                     590
535                                     591
536                                     592
537                                     593
538                                     594
539                                     595
540                                     596
541                                     597
542                                     598
543                                     599
544                                     600
545                                     601
546                                     602
547                                     603
548                                     604
549                                     605
550                                     606
551                                     607
552                                     608
553                                     609
554                                     610
555                                     611
556                                     612
557                                     613
558                                     614
559                                     615
560                                     616
561                                     617
562                                     618
563                                     619
564                                     620
565                                     621
566                                     622
567                                     623
568                                     624
569                                     625
570                                     626
571                                     627
572                                     628
573                                     629
574                                     630
575                                     631
576                                     632
577                                     633
578                                     634
579                                     635
580                                     636
581                                     637
582                                     638
583                                     639
584                                     640
585                                     641
586                                     642
587                                     643
588                                     644
589                                     645
590                                     646
591                                     647
592                                     648
593                                     649
594                                     650
595                                     651
596                                     652
597                                     653
598                                     654
599                                     655
600                                     656
601                                     657
602                                     658
603                                     659
604                                     660
605                                     661
606                                     662
607                                     663
608                                     664
609                                     665
610                                     666
611                                     667
612                                     668
613                                     669
614                                     670
615                                     671
616                                     672
617                                     673
618                                     674
619                                     675
620                                     676
621                                     677
622                                     678
623                                     679
624                                     680
625                                     681
626                                     682
627                                     683
628                                     684
629                                     685
630                                     686
631                                     687
632                                     688
633                                     689
634                                     690
635                                     691
636                                     692
637                                     693
638                                     694
639                                     695
640                                     696
641                                     697
642                                     698
643                                     699
644                                     700
645                                     701
646                                     702
647                                     703
648                                     704
649                                     705
650                                     706
651                                     707
652                                     708
653                                     709
654                                     710
655                                     711
656                                     712
657                                     713
658                                     714
659                                     715
660                                     716
661                                     717
662                                     718
663                                     719
664                                     720
665                                     721
666                                     722
667                                     723
668                                     724
669                                     725
670                                     726
671                                     727
672                                     728
673                                     729
674                                     730
675                                     731
676                                     732
677                                     733
678                                     734
679                                     735
680                                     736
681                                     737
682                                     738
683                                     739
684                                     740
685                                     741
686                                     742
687                                     743
688                                     744
689                                     745
690                                     746
691                                     747
692                                     748
693                                     749
694                                     750
695                                     751
696                                     752
697                                     753
698                                     754
699                                     755
700                                     756
701                                     757
702                                     758
703                                     759
704                                     760
705                                     761
706                                     762
707                                     763
708                                     764
709                                     765
710                                     766
711                                     767
712                                     768
713                                     769
714                                     770
715                                     771
716                                     772
717                                     773
718                                     774
719                                     775
720                                     776
721                                     777
722                                     778
723                                     779
724                                     780
725                                     781
726                                     782
727                                     783
728                                     784
729                                     785
730                                     786
731                                     787
732                                     788
733                                     789
734                                     790
735                                     791
736                                     792
737                                     793
738                                     794
739                                     795
740                                     796
741                                     797
742                                     798
743                                     799
744                                     800
745                                     801
746                                     802
747                                     803
748                                     804
749                                     805
750                                     806
751                                     807
752                                     808
753                                     809
754                                     810
755                                     811
756                                     812
757                                     813
758                                     814
759                                     815
760                                     816
761                                     817
762                                     818
763                                     819
764                                     820
765                                     821
766                                     822
767                                     823
768                                     824
769                                     825
770                                     826
771                                     827
772                                     828
773                                     829
774                                     830
775                                     831
776                                     832
777                                     833
778                                     834
779                                     835
780                                     836
781                                     837
782                                     838
783                                     839
784                                     840
785                                     841
786                                     842
787                                     843
788                                     844
789                                     845
790                                     846
791                                     847
792                                     848
793                                     849
794                                     850
795                                     851
796                                     852
797                                     853
798                                     854
799                                     855
800                                     856
801                                     857
802                                     858
803                                     859
804                                     860
805                                     861
806                                     862
807                                     863
808                                     864
809                                     865
810                                     866
811                                     867
812                                     868
813                                     869
814                                     870
815                                     871
816                                     872
817                                     873
818                                     874
819                                     875
820                                     876
821                                     877
822                                     878
823                                     879
824                                     880
825                                     881
826                                     882
827                                     883
828                                     884
829                                     885
830                                     886
831                                     887
832                                     888
833                                     889
834                                     890
835                                     891
836                                     892
837                                     893
838                                     894
839                                     895
840                                     896
841                                     897
842                                     898
843                                     899
844                                     900
845                                     901
846                                     902
847                                     903
848                                     904
849                                     905
850                                     906
851                                     907
852                                     908
853                                     909
854                                     910
855                                     911
856                                     912
857                                     913
858                                     914
859                                     915
860                                     916
861                                     917
862                                     918
863                                     919
864                                     920
865                                     921
866                                     922
867                                     923
868                                     924
869                                     925
870                                     926
871                                     927
872                                     928
873                                     929
874                                     930
875                                     931
876                                     932
877                                     933
878                                     934
879                                     935
880                                     936
881                                     937
882                                     938
883                                     939
884                                     940
885                                     941
886                                     942
887                                     943
888                                     944
889                                     945
890                                     946
891                                     947
892                                     948
893                                     949
894                                     950
895                                     951
896                                     952
897                                     953
898                                     954
899                                     955
900                                     956
901                                     957
902                                     958
903                                     959
904                                     960
905                                     961
906                                     962
907                                     963
908                                     964
909                                     965
910                                     966
911                                     967
912                                     968
913                                     969
914                                     970
915                                     971
916                                     972
917                                     973
918                                     974
919                                     975
920                                     976
921                                     977
922                                     978
923                                     979
924                                     980
925                                     981
926                                     982
927                                     983
928                                     984
929                                     985
930                                     986
931                                     987
932                                     988
933                                     989
934                                     990
935                                     991
936                                     992
937                                     993
938                                     994
939                                     995
940                                     996
941                                     997
942                                     998
943                                     999
944                                     1000
945                                     1001
946                                     1002
947                                     1003
948                                     1004
949                                     1005
950                                     1006
951                                     1007
952                                     1008
953                                     1009
954                                     1010
955                                     1011
956                                     1012
957                                     1013
958                                     1014
959                                     1015
960                                     1016
961                                     1017
962                                     1018
963                                     1019
964                                     1020
965                                     1021
966                                     1022
967                                     1023
968                                     1024
969                                     1025
970                                     1026
971                                     1027
972                                     1028
973                                     1029
974                                     1030
975                                     1031
976                                     1032
977                                     1033
978                                     1034
979                                     1035
980                                     1036
981                                     1037
982                                     1038
983                                     1039
984                                     1040
985                                     1041
986                                     1042
987                                     1043
988                                     1044
989                                     1045
990                                     1046
991                                     1047
992                                     1048
993                                     1049
994                                     1050
995                                     1051
996                                     1052
997                                     1053
998                                     1054
999                                     1055
1000                                    1056
1001                                    1057
1002                                    1058
1003                                    1059
1004                                    1060
1005                                    1061
1006                                    1062
1007                                    1063
1008                                    1064
1009                                    1065
1010                                    1066
1011                                    1067
1012                                    1068
1013                                    1069
1014                                    1070
1015                                    1071
1016                                    1072
1017                                    1073
1018                                    1074
1019                                    1075
1020                                    1076
1021                                    1077
1022                                    1078
1023                                    1079
1024                                    1080
1025                                    1081
1026                                    1082
1027                                    1083
1028                                    1084
1029                                    1085
1030                                    1086
1031                                    1087
1032                                    1088
1033                                    1089
1034                                    1090
1035                                    1091
1036                                    1092
1037                                    1093
1038                                    1094
1039                                    1095
1040                                    1096
1041                                    1097
1042                                    1098
1043                                    1099
1044                                    1100
1045                                    1101
1046                                    1102
1047                                    1103
1048                                    1104
1049                                    1105
1050                                    1106
1051                                    1107
1052                                    1108
1053                                    1109
1054                                    1110
1055                                    1111
1056                                    1112
1057                                    1113
1058                                    1114
1059                                    1115
1060                                    1116
1061                                    1117
1062                                    1118
1063                                    1119
1064                                    1120
1065                                    1121
1066                                    1122
1067                                    1123
1068                                    1124
1069                                    1125
1070                                    1126
1071                                    1127
1072                                    1128
1073                                    1129
1074                                    1130
1075                                    1131
1076                                    1132
1077                                    1133
1078                                    1134
1079                                    1135
1080                                    1136
1081                                    1137
1082                                    1138
1083                                    1139
1084                                    1140
1085                                    1141
1086                                    1142
1087                                    1143
1088                                    1144
1089                                    1145
1090                                    1146
1091                                    1147
1092                                    1148
1093                                    1149
1094                                    1150
1095                                    1151
1096                                    1152
1097                                    1153
1098                                    1154
1099                                    1155
1100                                    1156
1101                                    1157
1102                                    1158
1103                                    1159
1104                                    1160
1105                                    1161
1106                                    1162
1107                                    1163
1108                                    1164
1109                                    1165
1110                                    1166
1111                                    1167
1112                                    1168
1113                                    1169
1114                                    1170
1115                                    1171
1116                                    1172
1117                                    1173
1118                                    1174
1119                                    1175
1120                                    1176
1121                                    1177
1122                                    1178
1123                                    1179
1124                                    1180
1125                                    1181
1126                                    1182
1127                                    1183
1128                                    1184
1129                                    1185
1130                                    1186
1131                                    1187
1132                                    1188
1133                                    1189
1134                                    1190
1135                                    1191
1136                                    1192
1137                                    1193
1138                                    1194
1139                                    1195
1140                                    1196
1141                                    1197
1142                                    1198
1143                                    1199
1144                                    1200
1145                                    1201
1146                                    1202
1147                                    1203
1148                                    1204
1149                                    1205
1150                                    1206
1151                                    1207
1152                                    1208
1153                                    1209
1154                                    1210
1155                                    1211
1156                                    1212
1157                                    1213
1158                                    1214
1159                                    1215
1160                                    1216
1161                                    1217
1162                                    1218
1163                                    1219
1164                                    1220
1165                                    1221
1166                                    1222
1167                                    1223
1168                                    1224
1169                                    1225
1170                                    1226
1171                                    1227
1172                                    1228
1173                                    1229
1174                                    1230
1175                                    1231
1176                                    1232
1177                                    1233
1178                                    1234
1179                                    1235
1180                                    1236
1181                                    1237
1182                                    1238
1183                                    1239
1184                                    1240
1185                                    1241
1186                                    1242
1187                                    1243
1188                                    1244
1189                                    1245
1190                                    1246
1191                                    1247
1192                                    1248
1193                                    1249
1194                                    1250
1195                                    1251
1196                                    1252
1197                                    1253
1198                                    1254
1199                                    1255
1200                                    1256
1201                                    1257
1202                                    1258
1203                                    1259
1204                                    1260
1205                                    1261
1206                                    1262
1207                                    1263
1208                                    1264
1209                                    1265
1210                                    1266
1211                                    1267
1212                                    1268
1213                                    1269
1214                                    1270
1215                                    1271
1216                                    1272
1217                                    1273
1218                                    1274
1219                                    1275
1220                                    1276
1221                                    1277
1222                                    1278
1223                                    1279
1224                                    1280
1225                                    1281
1226                                    1282
1227                                    1283
1228                                    1284
1229                                    1285
1230                                    1286
1231                                    1287
1232                                    1288
1233                                    1289
1234                                    1290
1235                                    1291
1236                                    1292
1237                                    1293
1238                                    1294
1239                                    1295
1240                                    1296
1241                                    1297
1242                                    1298
1243                                    1299
1244                                    1300
1245                                    1301
1246                                    1302
1247                                    1303
1248                                    1304
1249                                    1305
1250                                    1306
1251                                    1307
1252                                    1308
1253                                    1309
1254                                    1310
1255                                    1311
1256                                    1312
1257                                    1313
1258                                    1314
1259                                    1315
1260                                    1316
1261                                    1317
1262                                    1318
1263                                    1319
1264                                    1320
1265                                    1321
1266                                    1322
1267                                    1323
1268                                    1324
1269                                    1325
1270                                    1326
1271                                    1327
1272                                    1328
1273                                    1329
1274                                    1330
1275                                    1331
1276                                    1332
1277                                    1333
1278                                    1334
1279                                    1335
1280                                    1336
1281                                    1337
1282                                    1338
1283                                    1339
1284                                    1340
1285                                    1341
1286                                    1342
1287                                    1343
1288                                    1344
1289                                    1345
1290                                    1346
1291                                    1347
1292                                    1348
1293                                    1349
1294                                    1350
1295                                    1351
1296                                    1352
1297                                    1353
1298                                    1354
1299                                    1355
1300                                    1356
1301                                    1357
1302                                    1358
1303                                    1359
1304                                    1360
1305                                    1361
1306                                    1362
1307                                    1363
1308                                    1364
1309                                    1365
1310                                    1366
1311                                    1367
1312                                    1368
1313                                    1369
1314                                    1370
1315                                    1371
1316                                    1372
1317                                    1373
1318                                    1374
1319                                    1375
1320                                    1376
1321                                    1377
1322                                    1378
1323                                    1379
1324                                    1380
1325                                    1381
1326                                    1382
1327                                    1383
1328                                    1384
1329                                    1385
1330                                    1386
1331                                    1387
1332                                    1388
1333                                    1389
1334                                    1390
1335                                    1391
1336                                    1392
1337                                    1393
1338                                    1394
1339                                    1395
1340                                    1396
1341                                    1397
1342                                    1398
1343                                    1399
1344                                    1400
1345                                    1401
1346                                    1402
1347                                    1403
1348                                    1404
1349                                    1405
1350                                    1406
1351                                    1407
1352                                    1408
1353                                    1409
1354                                    1410
1355                                    1411
1356                                    1412
1357                                    1413
1358                                    1414
1359                                    1415
1360                                    1416
1361                                    1417
1362                                    1418
1363                                    1419
1364                                    1420
1365                                    1421
1366                                    1422
1367                                    1423
1368                                    1424
1369                                    1425
1370                                    1426
1371                                    1427
1372                                    1428
1373                                    1429
1374                                    1430
1375                                    1431
1376                                    1432
1377                                    1433
1378                                    1434
1379                                    1435
1380                                    1436
1381                                    1437
1382                                    1438
1383                                    1439
1384                                    1440
1385                                    1441
1386                                    1442
1387                                    1443
1388                                    1444
1389                                    1445
1390                                    1446
1391                                    1447
1392                                    1448
1393                                    1449
1394                                    1450
1395                                    1451
1396                                    1452
1397                                    1453
1398                                    1454
1399                                    1455
1400                                    1456
1401                                    1457
1402                                    1458
1403                                    1459
1404                                    1460
1405                                    1461
1406                                    1462
1407                                    1463
1408                                    1464
1409                                    1465
1410                                    1466
1411                                    1467
1412                                    1468
1413                                    1469
1414                                    1470
1415                                    1471
1416                                    1472
1417                                    1473
1418                                    1474
1419                                    1475
1420                                    1476
1421                                    1477
1422                                    1478
1423                                    1479
1424                                    1480
1425                                    1481
1426                                    1482
1427                                    1483
1428                                    1484
1429                                    1485
1430                                    1486
1431                                    1487
1432                                    1488
1433                                    1489
1434                                    1490
1435                                    1491
1436                                    1492
1437                                    1493
1438                                    1494
1
```

```

333 epsilon_f = self.epsilon_0 / (1 + 370
        beta * n)  ## DECAYING EPSILON
334 Done = 0
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
    # sample a batch of
    # transitions
    transactions = self.
        replay_memory.sample(
            self.batch_size)
    # turn list of transactions
    # into transaction of
    # lists
    batch = Transition(*zip(*
        transactions))
    # backward step and
    # parameter update
    self.backward_dqn(batch,
        self.gamma)
    # update Q values indirectly by
    # updating the weights and
    # biases directly
    if Done==1: # THE EPISODE HAS
        ENDED, UPDATE...BE CAREFUL,
        THIS IS THE LAST STEP OF THE
        EPISODE
        if (self.method == "
            qlearning") or (self.
                method == "sarsa"):
            # compute gradients and
            # update weights
            self.backward(R, X,
                Qvalues, None, a1,
                h1, a2, None,
                future_reward=False,
                action_binary_mask=
                    A_binary_mask)
            # store history
            # todo: record max possible
            # reward per episode
            self.R_history[n] = np.copy(
                R) # reward per episode
            self.N_moves_history[n] = np
                .copy(i) # nr moves per
                episode
            # store norm of gradients
            self.dL_dW1_norm_history[n]
                = np.linalg.norm(self.
                    dL_dW1)
            self.dL_dW2_norm_history[n]
                = np.linalg.norm(self.
                    dL_dW2)
            # compute exponential moving
            # average (EMA) to
            # display during training
            ema = alpha*R + (1-alpha)*
                ema_previous
            if n == 0: # first episode
                ema = R
            ema_previous = ema
            if run_number is not None:
                episodes.set_description
                    (f"Run = {run_number
                        }; EMA Reward = {ema
                            :.2f}")
            else:
                episodes.set_description
                    (f"EMA Reward = {ema
                        :.2f}")
            break
    while Done==0:
        ##
        START THE EPISODE
        if (self.method == "qlearning"
            or (self.method == "dqn")):
            # compute Q values for the
            # given state
            a1, h1, a2, Qvalues = self.
                forward(X) # -> shape
                (O, 1)
            # choose an action A using
            # epsilon-greedy policy
            A_binary_mask, A_ind =
                EpsilonGreedy_Policy(
                    Qvalues, allowed_a,
                    epsilon_f) # -> shape
                    (O, 1)
            # take action and observe reward
            R and state S_prime
            S_prime, X_prime,
                allowed_a_prime, R, Done =
                    env.OneStep(A_ind)
            X_prime = np.expand_dims(X_prime,
                axis=1)
            X_prime = np.copy(np.vstack((np.
                array([[1]]), X_prime)))
            # add bias term
            n_steps += 1
            if self.method == "dqn":
                # store the transition in
                # memory
                self.replay_memory.push(X,
                    A_ind, R, X_prime, Done)

```



```

407                                     447             self.W1_target = np.copy(
408         else: # IF THE EPISODE IS NOT        self.W1)
409         OVER...                           448             self.W2_target = np.copy(
410                                     449             self.W2)
411         if self.method == "qlearning":      450
412             # chose next action off policy    451
413             Q_prime = np.max(self.           452
414             forward(X_prime)                 453
415             [-1])                           454
416                                     455
417         elif self.method == "sarsa":         456
418             # chose next action on policy     457
419             a1_prime, h1_prime,             458
420             a2_prime,                       459
421             Qvalues_prime = self.           460
422             .forward(X_prime)              461
423             -> shape (N_a, 1)              462
424                                     463
425             # chose next action and save it  464
426             A_binary_mask_prime,          465
427             A_ind_prime =                  466
428             EpsilonGreedy_Policy(         467
429             (Qvalues_prime,               468
430             allowed_a_prime,              469
431             epsilon_f))                   470
432                                     471
433             # get Qvalue of next action      472
434             Q_prime = Qvalues_prime[       473
435             A_ind_prime]                  474
436                                     475
437         if (self.method == "qlearning") or  476
438         method == "sarsa"):                477
439             # backpropagation and weight    478
440             update                         479
441             self.backward(R, X,            480
442             Qvalues, Q_prime, a1,          481
443             , h1, a2, self.gamma,         482
444             , future_reward=True,         483
445             , action_binary_mask         484
446             =A_binary_mask)              485
447                                     486
448         # NEXT STATE AND CO. BECOME        487
449         ACTUAL STATE...                   488
450         if self.method == "sarsa":         489
451             A_binary_mask = np.copy(A_     490
452             binary_mask_prime)            491
453             A_ind = np.copy(A_ind_prime)   492
454             a1 = np.copy(a1_prime)         493
455             h1 = np.copy(h1_prime)         494
456             a2 = np.copy(a2_prime)         495
457             Qvalues = np.copy(Qvalues_     496
458             prime)
459             S = np.copy(S_prime)           497
460             X = np.copy(X_prime)           498
461             allowed_a = np.copy(allowed_   499
462             a_prime)
463                                     500
464             i += 1 # UPDATE COUNTER FOR    501
465             NUMBER OF ACTIONS             502
466                                     503
467         if (self.method == "dqn") and      504
468         n_steps % self.C == 0):           505
469             # update target network        506
470             every C steps

```

```

471                                     507
472         training_end = time.time()
473         self.training_time_in_seconds =
474         training_end - training_start
475
476         return None
477
478     except KeyboardInterrupt as e:
479         # return nothing
480         training_end = time.time()
481         self.training_time_in_seconds =
482         training_end - training_start
483
484         return None
485
486     def save(self, name_extension=None):
487         # create directory for the model
488         name = f"{self.method}_{self.act_f_1_name}_{
489         self.act_f_2_name}"
490         if name_extension is not None:
491             name += f"_{name_extension}"
492
493         path = f"models/{name}"
494         if not os.path.isdir(path): os.mkdir(path)
495         print(f"saving to: {path}")
496
497         # save weights
498         np.save(f"{path}/W1.npy", self.W1)
499         np.save(f"{path}/W2.npy", self.W2)
500
501         # save training history
502         np.save(f"{path}/training_history_R.npy",
503         self.R_history)
504         np.save(f"{path}/training_history_N_moves.
505         npy", self.N_moves_history)
506         np.save(f"{path}/
507         training_history_dL_dW1_norm.npy", self.
508         dL_dW1_norm_history)
509         np.save(f"{path}/
510         training_history_dL_dW2_norm.npy", self.
511         dL_dW2_norm_history)
512
513         # save training parameters and other general
514         info
515         params = {
516             "method": self.method,
517             "N_episodes": self.N_episodes,
518             "eta": self.eta,
519             "epsilon_0": self.epsilon_0,
520             "beta": self.beta,
521             "gamma": self.gamma,
522             "alpha": self.alpha,
523             # "gradient_clip": self.gradient_clip,
524             "seed": self.seed,
525             "D": self.D,
526             "K": self.K,
527             "O": self.O,
528             "training_time_in_seconds": self.
529             training_time_in_seconds
530         }
531         if self.method == "dqn":
532             params["capacity"] = self.capacity
533             params["batch_size"] = self.batch_size
534             params["C"] = self.C
535         with open(f"{path}/training_parameters.json",
536         "w") as f:
537             json.dump(params, f)

```

```

507
508
509 def load_from(method, act_f_1, act_f_2,
    name_extension=None):
510
511     # read values and store in neural network
    instance
512     name = f"{method}_{act_f_1}_{act_f_2}"
513     if name_extension is not None:
514         name += f"_{name_extension}"
515
516     path = f"models/{name}"
517     # print(f"loading from: {path}")
518
519     # initialize neural network
520     nn = NeuralNetwork(0,0,0, activation_function_1=
        act_f_1, activation_function_2=act_f_2,
        method=method)
521
522     # network weights
523     nn.W1 = np.load(f"{path}/W1.npy")
524     nn.W2 = np.load(f"{path}/W2.npy")
525
526     # network training history
527     nn.R_history = np.load(f"{path}/
        training_history_R.npy")
528     nn.N_moves_history = np.load(f"{path}/
        training_history_N_moves.npy")
529     nn.dL_dW1_norm_history = np.load(f"{path}/
        training_history_dL_dW1_norm.npy")
530     nn.dL_dW2_norm_history = np.load(f"{path}/
        training_history_dL_dW2_norm.npy")
531
532     # network training parameters
533     with open(f"{path}/training_parameters.json", "
        ") as f:
534         params = json.load(f)
535
536     # set parameters to the network instance
537     nn.method = params["method"]
538     nn.N_episodes = int(params["N_episodes"])
539     nn.eta = float(params["eta"])
540     nn.epsilon_0 = float(params["epsilon_0"])
541     nn.beta = float(params["beta"])
542     nn.gamma = float(params["gamma"])
543     nn.alpha = float(params["alpha"])
544     # nn.gradient_clip = float(params["
        gradient_clip"])
545     try:
546         nn.seed = int(params["seed"])
547     except:
548         nn.seed = params["seed"]
549     nn.D = int(params["D"])
550     nn.K = int(params["K"])
551     nn.O = int(params["O"])
552     nn.training_time_in_seconds = float(params["
        training_time_in_seconds"])
553
554     if nn.method == "dqn":
555         nn.capacity = int(params["capacity"])
556         nn.batch_size = int(params["batch_size"
            ])
557         nn.C = int(params["C"])
558
559     if nn.method == "dqn":
560         nn.W1_target = np.copy(nn.W1)
561         nn.W2_target = np.copy(nn.W2)
562
563     return nn
564

```

Listing 1: Object oriented implementation of the neural networks, which can be instantiated with specifications for the model architecture and a method: “sarsa”, “qlearning” or “dqn”. The training loop will adapt automatically.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def moving_average(a, n=3) :
5     steps = len(a)-n
6     ma = np.full(steps, np.nan)
7     for i in range(steps):
8         ma[i] = np.mean(a[i:i+n])
9     return ma, np.arange(steps)
10
11
12 def exponential_moving_average(array, alpha=0.001):
13     """
14     Calculate exponential moving average of an array
15     """
16     ema = np.full(len(array), np.nan)
17     ema[0] = array[0]
18     for i in range(1, len(array)):
19         ema[i] = alpha * array[i] + (1 - alpha) *
            ema[i-1]
20     return ema
21
22
23 def save_avg_statistics(histories, method):
24     # unpack histories
25     R_histories = [history[0] for history in
        histories]
26     N_moves_histories = [history[1] for history in
        histories]
27     training_times = [history[2] for history in
        histories]
28     layer1_gradient_norms_histories = [history[3]
        for history in histories]
29     layer2_gradient_norms_histories = [history[4]
        for history in histories]
30
31     # turn into numpy arrays
32     R_histories = np.hstack(R_histories)
33     N_moves_histories = np.hstack(N_moves_histories)
34     training_times = np.hstack(training_times)
35     layer1_gradient_norms_histories = np.hstack(
        layer1_gradient_norms_histories)
36     layer2_gradient_norms_histories = np.hstack(
        layer2_gradient_norms_histories)
37
38     # compute mean and standard deviation for each
        row of the histories
39     R_mean = np.mean(R_histories, axis=1)
40     R_std = np.std(R_histories, axis=1)
41
42     N_moves_mean = np.mean(N_moves_histories, axis
        =1)
43     N_moves_std = np.std(N_moves_histories, axis=1)
44
45     layer1_gradient_norms_mean = np.mean(
        layer1_gradient_norms_histories, axis=1)
46     layer1_gradient_norms_std = np.std(
        layer1_gradient_norms_histories, axis=1)
47
48     layer2_gradient_norms_mean = np.mean(
        layer2_gradient_norms_histories, axis=1)
49     layer2_gradient_norms_std = np.std(
        layer2_gradient_norms_histories, axis=1)
50
51     # save to file

```

```

52     np.save(f"statistics/{method}_R_mean.npy",
53             R_mean)
54     np.save(f"statistics/{method}_R_std.npy", R_std)
55     np.save(f"statistics/{method}_N_moves_mean.npy",
56             N_moves_mean)
57     np.save(f"statistics/{method}_N_moves_std.npy",
58             N_moves_std)
59     np.save(f"statistics/{method}_training_times.npy",
60             training_times)
61     np.save(f"statistics/{method}_layer1_gradient_norms_mean.npy",
62             layer1_gradient_norms_mean)
63     np.save(f"statistics/{method}_layer1_gradient_norms_std.npy",
64             layer1_gradient_norms_std)
65     np.save(f"statistics/{method}_layer2_gradient_norms_mean.npy",
66             layer2_gradient_norms_mean)
67     np.save(f"statistics/{method}_layer2_gradient_norms_std.npy",
68             layer2_gradient_norms_std)
69
70 def load_avg_statistics(method):
71     R_mean = np.load(f"statistics/{method}_R_mean.npy")
72     R_std = np.load(f"statistics/{method}_R_std.npy")
73
74     N_moves_mean = np.load(f"statistics/{method}_N_moves_mean.npy")
75     N_moves_std = np.load(f"statistics/{method}_N_moves_std.npy")
76
77     training_times = np.load(f"statistics/{method}_training_times.npy")
78
79     layer1_gradient_norms_mean = np.load(f"statistics/{method}_layer1_gradient_norms_mean.npy")
80     layer1_gradient_norms_std = np.load(f"statistics/{method}_layer1_gradient_norms_std.npy")
81
82     layer2_gradient_norms_mean = np.load(f"statistics/{method}_layer2_gradient_norms_mean.npy")
83     layer2_gradient_norms_std = np.load(f"statistics/{method}_layer2_gradient_norms_std.npy")
84
85     return R_mean, R_std, N_moves_mean, N_moves_std, training_times, layer1_gradient_norms_mean, layer1_gradient_norms_std, layer2_gradient_norms_mean, layer2_gradient_norms_std
86
87 def printable_name(method):
88     if method == "sarsa":
89         return "SARSA"
90     elif method == "qlearning":
91         return "Q-Learning"
92     elif method == "dqn":
93         return "DQN"
94     else:
95         return None

```

Listing 2: Helper functions used throughout the implementation of the neural network and the notebooks, where the experiments were conducted.