

# Chess Assignment

Introduction to Reinforcement Learning (Spring 2022). Source code available at: [https://github.com/TwoDigitsOneNumber/IntroRL\\_ChessAssignment](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 DQN<sup>1</sup>. 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  $\beta$  and  $\gamma$ , 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.

<sup>1</sup>SARSA 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-Learning<sup>2</sup>

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  $\gamma$  (see Section III-C). Both algorithms repeatedly choose and take actions in the environment according to some policy  $\pi$ , e.g. an  $\epsilon$ -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  $\pi$ , 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  $\pi$ .

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

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

<sup>2</sup>Answer 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 Replay<sup>3</sup>

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)<sup>4</sup>

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 runs<sup>5</sup> 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  $\beta$  and  $\gamma$ <sup>6</sup> we trained 49 agents with different combinations for  $\beta$  and  $\gamma$  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-Learning<sup>7</sup>) and Mnih et al. [3] (DQN<sup>8</sup>). For the implementation see file `neural_net.py`

<sup>3</sup>Answer to “group only” task 2.

<sup>4</sup>Answer to task 5: Describing the used method.

<sup>5</sup>Answers to task 3 and 5.

<sup>6</sup>Answer to task 4.

<sup>7</sup>SARSA as answer to task 3 and Q-Learning as additional algorithm.

<sup>8</sup>Answer 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 epsilon $\epsilon_0$	0.2
Learning rate $\eta$	0.035
Decay rate of $\epsilon$ , $\beta$	0.00005
Discount factor $\gamma$	0.85

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

### III. RESULTS

#### A. Seeded Runs<sup>9</sup>

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  $\epsilon$  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.

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

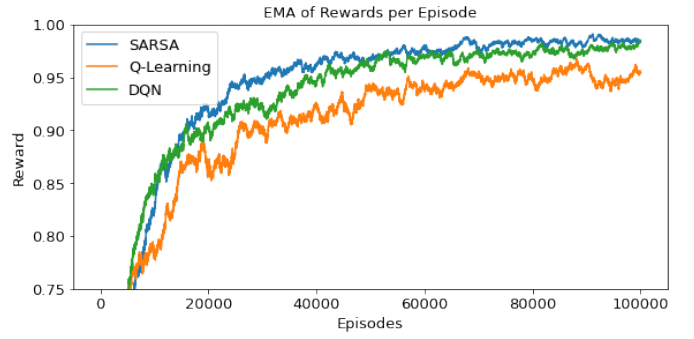


Fig. 1: Exponential moving average of the rewards achieved during training for 100000 episodes with identical hyper-parameters, weight initialization and model architecture.

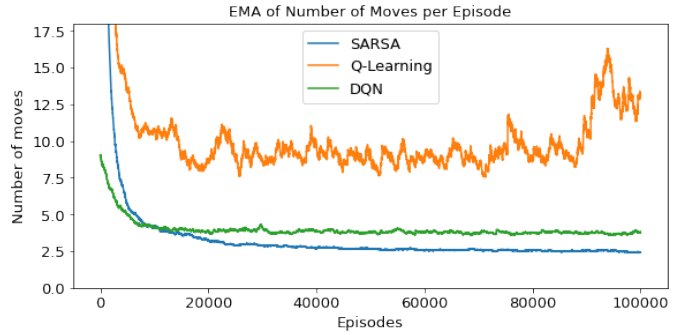


Fig. 2: Exponential moving average of the number of moves per episode achieved during training for 100000 episodes with identical hyper-parameters, weight initialization and model architecture.

As suggested by Mnih et al. [3], [5], DQN<sup>10</sup> 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

<sup>10</sup>Answer 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-parameters<sup>11</sup>

Figure 3(a) depicts the rewards and number of moves per episode as a function of  $\beta$  and  $\gamma$ . We can see that the reward increases monotonically as  $\gamma$  is increased, suggesting that a value of  $\gamma \in [0.80, 1)$  should be chosen for almost all values of  $\beta$ . 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  $\gamma$  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  $\beta$  and the rewards, apart from  $\beta = 0$  being an inferior choice for all values of  $\gamma$ . In Figure 3(b) we can see that the number of steps taken by the agent decreases drastically when increasing  $\gamma$  from very low levels, but this effect seems larger for larger values of  $\beta$ . 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  $\gamma$  the choice of  $\beta$  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

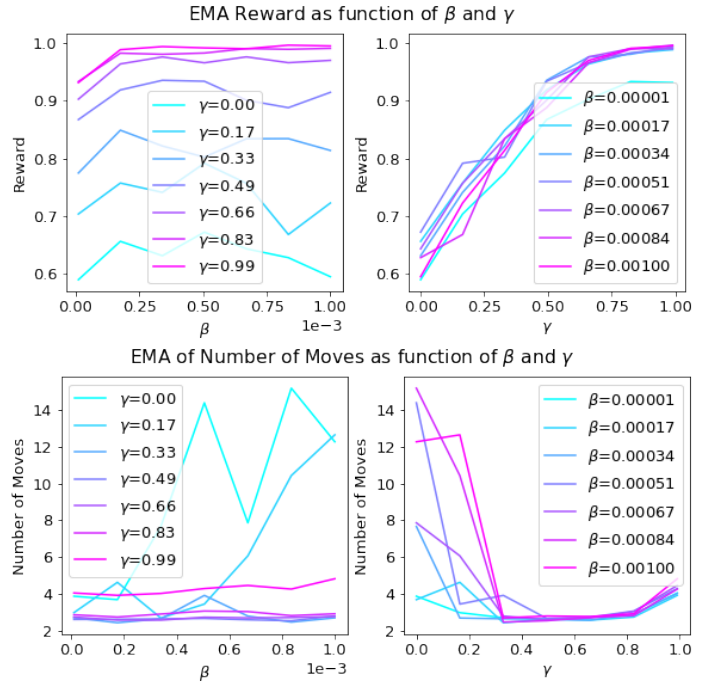


Fig. 3: Rewards and number of moves as functions of the speed of the decaying trend  $\beta$  and the discount factor  $\gamma$  after training a SARSA agent for 40000 episodes.

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.

<sup>11</sup> 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](https://github.com/TwoDigitsOneNumber/IntroRL_ChessAssignment). Along this we provide a conda environment 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 shapeensions 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         act_f_and_gradient(activation_function_1)
160         self.act_f_2, self.grad_act_f_2 =
161         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                                     276
217 # -> shape (O, 1)                  277
218 else: # immediate reward            278
219     delta = R_rep - Qvalues # -> shape (O
220                                     279
221                                     280
222                                     281
223 # update only action that was taken, i.e. 282
224 all rows apart from the one          283
225 corresponding to the action taken (    284
226 action index) are 0                  285
227 delta = delta*action_binary_mask      286
228                                     287
229 self.compute_gradients(delta, a1, h1, a2, 288
230 self.update_parameters(self.eta)      289
231                                     290
232 def backward_dqn(self, batch, gamma):  291
233     """                                292
234     backward for method "dqn"          293
235     """                                294
236                                     295
237 # ===== compute targets y and feature matrix 296
238 X =====                            297
239                                     298
240 # turn batch into individual tuples, numpy 299
241 arrays, or lists                     300
242 states = batch.state                 301
243 rewards = np.array(list(batch.reward)) 302
244 actions = np.array(list(batch.action)) 303
245 next_states = list(batch.next_state)   304
246 dones = np.array(list(batch.done))     305
247                                     306
248 # compute targets y and feature matrix X    307
249 y = np.zeros((self.O, len(dones)))      308
250 for j in np.arange(len(dones)):          309
251     if dones[j]: # if done, set y_j = r_j 310
252         y[actions[j], j] = rewards[j]    311
253     else:                                       312
254         # compute Q_prime                   313
255         Q_target = self.forward(next_states 314
256                                 j], target=True)[-1] 315
257         y[actions[j], j] = rewards[j] +    316
258         gamma*np.max(Q_target)            317
259                                     318
260                                     319
261 # convert states to feature matrix X        320
262 X = np.hstack((states))                  321
263                                     322
264                                     323
265 # ===== compute TD error (aka delta) ===== 324
266                                     325
267 al, h1, a2, Qvalues = self.forward(X)     326
268 delta = y - Qvalues # -> shape (O,      327
269 batch_size)                               328
270                                     329
271 self.compute_gradients(delta, a1, h1, a2, 330
272 self.update_parameters(self.eta)          331
273                                     332
274                                     333
275                                     334
276                                     335
277                                     336
278                                     337
279                                     338
280                                     339
281                                     340
282                                     341
283                                     342
284                                     343
285                                     344
286                                     345
287                                     346
288                                     347
289                                     348
290                                     349
291                                     350
292                                     351
293                                     352
294                                     353
295                                     354
296                                     355
297                                     356
298                                     357
299                                     358
300                                     359
301                                     360
302                                     361
303                                     362
304                                     363
305                                     364
306                                     365
307                                     366
308                                     367
309                                     368
310                                     369
311                                     370
312                                     371
313                                     372
314                                     373
315                                     374
316                                     375
317                                     376
318                                     377
319                                     378
320                                     379
321                                     380
322                                     381
323                                     382
324                                     383
325                                     384
326                                     385
327                                     386
328                                     387
329                                     388
330                                     389
331                                     390
332                                     391
333                                     392
334                                     393
335                                     394
336                                     395
337                                     396
338                                     397
339                                     398
340                                     399
341                                     400
342                                     401
343                                     402
344                                     403
345                                     404
346                                     405
347                                     406
348                                     407
349                                     408
350                                     409
351                                     410
352                                     411
353                                     412
354                                     413
355                                     414
356                                     415
357                                     416
358                                     417
359                                     418
360                                     419
361                                     420
362                                     421
363                                     422
364                                     423
365                                     424
366                                     425
367                                     426
368                                     427
369                                     428
370                                     429
371                                     430
372                                     431
373                                     432
374                                     433
375                                     434
376                                     435
377                                     436
378                                     437
379                                     438
380                                     439
381                                     440
382                                     441
383                                     442
384                                     443
385                                     444
386                                     445
387                                     446
388                                     447
389                                     448
390                                     449
391                                     450
392                                     451
393                                     452
394                                     453
395                                     454
396                                     455
397                                     456
398                                     457
399                                     458
400                                     459
401                                     460
402                                     461
403                                     462
404                                     463
405                                     464
406                                     465
407                                     466
408                                     467
409                                     468
410                                     469
411                                     470
412                                     471
413                                     472
414                                     473
415                                     474
416                                     475
417                                     476
418                                     477
419                                     478
420                                     479
421                                     480
422                                     481
423                                     482
424                                     483
425                                     484
426                                     485
427                                     486
428                                     487
429                                     488
430                                     489
431                                     490
432                                     491
433                                     492
434                                     493
435                                     494
436                                     495
437                                     496
438                                     497
439                                     498
440                                     499
441                                     500
442                                     501
443                                     502
444                                     503
445                                     504
446                                     505
447                                     506
448                                     507
449                                     508
450                                     509
451                                     510
452                                     511
453                                     512
454                                     513
455                                     514
456                                     515
457                                     516
458                                     517
459                                     518
460                                     519
461                                     520
462                                     521
463                                     522
464                                     523
465                                     524
466                                     525
467                                     526
468                                     527
469                                     528
470                                     529
471                                     530
472                                     531
473                                     532
474                                     533
475                                     534
476                                     535
477                                     536
478                                     537
479                                     538
480                                     539
481                                     540
482                                     541
483                                     542
484                                     543
485                                     544
486                                     545
487                                     546
488                                     547
489                                     548
490                                     549
491                                     550
492                                     551
493                                     552
494                                     553
495                                     554
496                                     555
497                                     556
498                                     557
499                                     558
500                                     559
501                                     560
502                                     561
503                                     562
504                                     563
505                                     564
506                                     565
507                                     566
508                                     567
509                                     568
510                                     569
511                                     570
512                                     571
513                                     572
514                                     573
515                                     574
516                                     575
517                                     576
518                                     577
519                                     578
520                                     579
521                                     580
522                                     581
523                                     582
524                                     583
525                                     584
526                                     585
527                                     586
528                                     587
529                                     588
530                                     589
531                                     590
532                                     591
533                                     592
534                                     593
535                                     594
536                                     595
537                                     596
538                                     597
539                                     598
540                                     599
541                                     600
542                                     601
543                                     602
544                                     603
545                                     604
546                                     605
547                                     606
548                                     607
549                                     608
550                                     609
551                                     610
552                                     611
553                                     612
554                                     613
555                                     614
556                                     615
557                                     616
558                                     617
559                                     618
560                                     619
561                                     620
562                                     621
563                                     622
564                                     623
565                                     624
566                                     625
567                                     626
568                                     627
569                                     628
570                                     629
571                                     630
572                                     631
573                                     632
574                                     633
575                                     634
576                                     635
577                                     636
578                                     637
579                                     638
580                                     639
581                                     640
582                                     641
583                                     642
584                                     643
585                                     644
586                                     645
587                                     646
588                                     647
589                                     648
590                                     649
591                                     650
592                                     651
593                                     652
594                                     653
595                                     654
596                                     655
597                                     656
598                                     657
599                                     658
600                                     659
601                                     660
602                                     661
603                                     662
604                                     663
605                                     664
606                                     665
607                                     666
608                                     667
609                                     668
610                                     669
611                                     670
612                                     671
613                                     672
614                                     673
615                                     674
616                                     675
617                                     676
618                                     677
619                                     678
620                                     679
621                                     680
622                                     681
623                                     682
624                                     683
625                                     684
626                                     685
627                                     686
628                                     687
629                                     688
630                                     689
631                                     690
632                                     691
633                                     692
634                                     693
635                                     694
636                                     695
637                                     696
638                                     697
639                                     698
640                                     699
641                                     700
642                                     701
643                                     702
644                                     703
645                                     704
646                                     705
647                                     706
648                                     707
649                                     708
650                                     709
651                                     710
652                                     711
653                                     712
654                                     713
655                                     714
656                                     715
657                                     716
658                                     717
659                                     718
660                                     719
661                                     720
662                                     721
663                                     722
664                                     723
665                                     724
666                                     725
667                                     726
668                                     727
669                                     728
670                                     729
671                                     730
672                                     731
673                                     732
674                                     733
675                                     734
676                                     735
677                                     736
678                                     737
679                                     738
680                                     739
681                                     740
682                                     741
683                                     742
684                                     743
685                                     744
686                                     745
687                                     746
688                                     747
689                                     748
690                                     749
691                                     750
692                                     751
693                                     752
694                                     753
695                                     754
696                                     755
697                                     756
698                                     757
699                                     758
700                                     759
701                                     760
702                                     761
703                                     762
704                                     763
705                                     764
706                                     765
707                                     766
708                                     767
709                                     768
710                                     769
711                                     770
712                                     771
713                                     772
714                                     773
715                                     774
716                                     775
717                                     776
718                                     777
719                                     778
720                                     779
721                                     780
722                                     781
723                                     782
724                                     783
725                                     784
726                                     785
727                                     786
728                                     787
729                                     788
730                                     789
731                                     790
732                                     791
733                                     792
734                                     793
735                                     794
736                                     795
737                                     796
738                                     797
739                                     798
740                                     799
741                                     800
742                                     801
743                                     802
744                                     803
745                                     804
746                                     805
747                                     806
748                                     807
749                                     808
750                                     809
751                                     810
752                                     811
753                                     812
754                                     813
755                                     814
756                                     815
757                                     816
758                                     817
759                                     818
760                                     819
761                                     820
762                                     821
763                                     822
764                                     823
765                                     824
766                                     825
767                                     826
768                                     827
769                                     828
770                                     829
771                                     830
772                                     831
773                                     832
774                                     833
775                                     834
776                                     835
777                                     836
778                                     837
779                                     838
780                                     839
781                                     840
782                                     841
783                                     842
784                                     843
785                                     844
786                                     845
787                                     846
788                                     847
789                                     848
790                                     849
791                                     850
792                                     851
793                                     852
794                                     853
795                                     854
796                                     855
797                                     856
798                                     857
799                                     858
800                                     859
801                                     860
802                                     861
803                                     862
804                                     863
805                                     864
806                                     865
807                                     866
808                                     867
809                                     868
810                                     869
811                                     870
812                                     871
813                                     872
814                                     873
815                                     874
816                                     875
817                                     876
818                                     877
819                                     878
820                                     879
821                                     880
822                                     881
823                                     882
824                                     883
825                                     884
826                                     885
827                                     886
828                                     887
829                                     888
830                                     889
831                                     890
832                                     891
833                                     892
834                                     893
835                                     894
836                                     895
837                                     896
838                                     897
839                                     898
840                                     899
841                                     900
842                                     901
843                                     902
844                                     903
845                                     904
846                                     905
847                                     906
848                                     907
849                                     908
850                                     909
851                                     910
852                                     911
853                                     912
854                                     913
855                                     914
856                                     915
857                                     916
858                                     917
859                                     918
860                                     919
861                                     920
862                                     921
863                                     922
864                                     923
865                                     924
866                                     925
867                                     926
868                                     927
869                                     928
870                                     929
871                                     930
872                                     931
873                                     932
874                                     933
875                                     934
876                                     935
877                                     936
878                                     937
879                                     938
880                                     939
881                                     940
882                                     941
883                                     942
884                                     943
885                                     944
886                                     945
887                                     946
888                                     947
889                                     948
890                                     949
891                                     950
892                                     951
893                                     952
894                                     953
895                                     954
896                                     955
897                                     956
898                                     957
899                                     958
900                                     959
901                                     960
902                                     961
903                                     962
904                                     963
905                                     964
906                                     965
907                                     966
908                                     967
909                                     968
910                                     969
911                                     970
912                                     971
913                                     972
914                                     973
915                                     974
916                                     975
917                                     976
918                                     977
919                                     978
920                                     979
921                                     980
922                                     981
923                                     982
924                                     983
925                                     984
926                                     985
927                                     986
928                                     987
929                                     988
930                                     989
931                                     990
932                                     991
933                                     992
934                                     993
935                                     994
936                                     995
937                                     996
938                                     997
939                                     998
940                                     999
941                                     1000
942                                     1001
943                                     1002
944                                     1003
945                                     1004
946                                     1005
947                                     1006
948                                     1007
949                                     1008
950                                     1009
951                                     1010
952                                     1011
953                                     1012
954                                     1013
955                                     1014
956                                     1015
957                                     1016
958                                     1017
959                                     1018
960                                     1019
961                                     1020
962                                     1021
963                                     1022
964                                     1023
965                                     1024
966                                     1025
967                                     1026
968                                     1027
969                                     1028
970                                     1029
971                                     1030
972                                     1031
973                                     1032
974                                     1033
975                                     1034
976                                     1035
977                                     1036
978                                     1037
979                                     1038
980                                     1039
981                                     1040
982                                     1041
983                                     1042
984                                     1043
985                                     1044
986                                     1045
987                                     1046
988                                     1047
989                                     1048
990                                     1049
991                                     1050
992                                     1051
993                                     1052
994                                     1053
995                                     1054
996                                     1055
997                                     1056
998                                     1057
999                                     1058
1000                                    1059
1001                                    1060
1002                                    1061
1003                                    1062
1004                                    1063
1005                                    1064
1006                                    1065
1007                                    1066
1008                                    1067
1009                                    1068
1010                                    1069
1011                                    1070
1012                                    1071
1013                                    1072
1014                                    1073
1015                                    1074
1016                                    1075
1017                                    1076
1018                                    1077
1019                                    1078
1020                                    1079
1021                                    1080
1022                                    1081
1023                                    1082
1024                                    1083
1025                                    1084
1026                                    1085
1027                                    1086
1028                                    1087
1029                                    1088
1030                                    1089
1031                                    1090
1032                                    1091
1033                                    1092
1034                                    1093
1035                                    1094
1036                                    1095
1037                                    1096
1038                                    1097
1039                                    1098
1040                                    1099
1041                                    1100
1042                                    1101
1043                                    1102
1044                                    1103
1045                                    1104
1046                                    1105
1047                                    1106
1048                                    1107
1049                                    1108
1050                                    1109
1051                                    1110
1052                                    1111
1053                                    1112
1054                                    1113
1055                                    1114
1056                                    1115
1057                                    1116
1058                                    1117
1059                                    1118
1060                                    1119
1061                                    1120
1062                                    1121
1063                                    1122
1064                                    1123
1065                                    1124
1066                                    1125
1067                                    1126
1068                                    1127
1069                                    1128
1070                                    1129
1071                                    1130
1072                                    1131
1073                                    1132
1074                                    1133
1075                                    1134
1076                                    1135
1077                                    1136
1078                                    1137
1079                                    1138
1080                                    1139
1081                                    1140
1082                                    1141
1083                                    1142
1084                                    1143
1085                                    1144
1086                                    1145
1087                                    1146
1088                                    1147
1089                                    1148
1090                                    1149
1091                                    1150
1092                                    1151
1093                                    1152
1094                                    1153
1095                                    1154
1096                                    1155
1097                                    1156
1098                                    1157
1099                                    1158
1100                                    1159
1101                                    1160
1102                                    1161
1103                                    1162
1104                                    1163
1105                                    1164
1106                                    1165
1107                                    1166
1108                                    1167
1109                                    1168
1110                                    1169
1111                                    1170
1112                                    1171
1113                                    1172
1114                                    1173
1115                                    1174
1116                                    1175
1117                                    1176
1118                                    1177
1119                                    1178
1120                                    1179
1121                                    1180
1122                                    1181
1123                                    1182
1124                                    1183
1125                                    1184
1126                                    1185
1127                                    1186
1128                                    1187
1129                                    1188
1130                                    1189
1131                                    1190
1132                                    1191
1133                                    1192
1134                                    1193
1135                                    1194
1136                                    1195
1137                                    1196
1138                                    1197
1139                                    1198
1140                                    1199
1141                                    1200
1142                                    1201
1143                                    1202
1144                                    1203
1145                                    1204
1146                                    1205
1147                                    1206
1148                                    1207
1149                                    1208
1150                                    1209
1151                                    1210
1152                                    1211
1153                                    1212
1154                                    1213
1155                                    1214
1156                                    1215
1157                                    1216
1158                                    1217
1159                                    1218
1160                                    1219
1161                                    1220
1162                                    1221
1163                                    1222
1164                                    1223
1165                                    1224
1166                                    1225
1167                                    1226
1168                                    1227
1169                                    1228
1170                                    1229
1171                                    1230
1172                                    1231
1173                                    1232
1174                                    1233
1175                                    1234
1176                                    1235
1177                                    1236
1178                                    1237
1179                                    1238
1180                                    1239
1181                                    1240
1182                                    1241
1183                                    1242
1184                                    1243
1185                                    1244
1186                                    1245
1187                                    1246
1188                                    1247
1189                                    1248
1190                                    1249
1191                                    1250
1192                                    1251
1193                                    1252
1194                                    1253
1195                                    1254
1196                                    1255
1197                                    1256
1198                                    1257
1199                                    1258
1200                                    1259
1201                                    1260
1202                                    1261
1203                                    1262
1204                                    1263
1205                                    1264
1206                                    1265
1207                                    1266
1208                                    1267
1209                                    1268
1210                                    1269
1211                                    1270
1212                                    1271
1213                                    1272
1214                                    1273
1215                                    1274
1216                                    1275
1217                                    1276
1218                                    1277
1219                                    1278
1220                                    1279
1221                                    1280
1222                                    1281
1223                                    1282
1224                                    1283
1225                                    1284
1226                                    1285
1227                                    1286
1228                                    1287
1229                                    1288
1230                                    1289
1231                                    1290
1232                                    1291
1233                                    1292
1234                                    1293
1235                                    1294
1236                                    1295
1237                                    1296
1238                                    1297
1239                                    1298
1240                                    1299
1241                                    1300
1242                                    1301
1243                                    1302
1244                                    1303
1245                                    1304
1246                                    1305
1247                                    1306
1248                                    1307
1249                                    1308
1250                                    1309
1251                                    1310
1252                                    1311
1253                                    1312
1254                                    1313
1255                                    1314
1256                                    1315
1257                                    1316
1258                                    1317
1259                                    1318
1260                                    1319
1261                                    1320
1262                                    1321
1263                                    1322
1264                                    1323
1265                                    1324
1266                                    1325
1267                                    1326
1268                                    1327
1269                                    1328
1270                                    1329
1271                                    1330
1272                                    1331
1273                                    1332
1274                                    1333
1275                                    1334
1276                                    1335
1277                                    1336
1278                                    1337
1279                                    1338
1280                                    1339
1281                                    1340
1282                                    1341
1283                                    1342
1284                                    1343
1285                                    1344
1286                                    1345
1287                                    1346
1288                                    1347
1289                                    1348
1290                                    1349
1291                                    1350
1292                                    1351
1293                                    1352
1294                                    1353
1295                                    1354
1296                                    1355
1297                                    1356
1298                                    1357
1299                                    1358
1300                                    1359
1301                                    1360
1302                                    1361
1303                                    1362
1304                                    1363
1305                                    1364
1306                                    1365
1307                                    1366
1308                                    1367
1309                                    1368
1310                                    1369
1311                                    1370
1312                                    1371
1313                                    1372
1314                                    1373
1315                                    1374
1316                                    1375
1317                                    1376
1318                                    1377
1319                                    1378
1320                                    1379
1321                                    1380
1322                                    1381
1323                                    1382
1324                                    1383
1325                                    1384
1326                                    1385
1327                                    1386
1328                                    1387
1329                                    1388
1330                                    1389
1331                                    1390
1332                                    1391
1333                                    1392
1334                                    1393
1335                                    1394
1336                                    1395
1337                                    1396
1338                                    1397
1339                                    1398
1340                                    1399
1341                                    1400
1342                                    1401
1343                                    1402
1344                                    1403
1345                                    1404
1346                                    1405
1347                                    1406
1348                                    1407
1349                                    1408
1350                                    1409
1351                                    1410
1352                                    1411
1353                                    1412
1354                                    1413
1355                                    1414
1356                                    1415
1357                                    1416
1358                                    1417
1359                                    1418
1360                                    1419
1361                                    1420
1362                                    1421
1363                                    1422
1364                                    1423
1365                                    1424
1366                                    1425
1367                                    1426
1368                                    1427
1369                                    1428
1370                                    1429
1371                                    1430
1372                                    1431
1373                                    1432
1374                                    1433
1375                                    1434
1376                                    1435
1377                                    1436
1378                                    1437
1379                                    1438
1380                                    1439
1381                                    1440
1382                                    1441
1383                                    1442
1384                                    1443
1385                                    1444
1386                                    1445
1387                                    1446
1388                                    1447
1389                                    1448
1390                                    1449
1391                                    1450
1392                                    1451
1393                                    1452
1394                                    1453
1395                                    1454
1396                                    1455
1397                                    1456
1398                                    1457
1399                                    1458
1400                                    1459
1401                                    1460
1402                                    1461
1403                                    1462
1404                                    1463
1405                                    1464
1406                                    1465
1407                                    1466
1408                                    1467
1409                                    1468
1410                                    1469
1411                                    1470
1412                                    1471
1413                                    1472
1414                                    1473
1415                                    1474
1416                                    1475
1417                                    1476
1418                                    1477
1419                                    1478
1420                                    1479
1421                                    1480
1422                                    1481
1423                                    1482
1424                                    1483
1425                                    1484
1426                                    1485
1427                                    1486
1428                                    1487
1429                                    1488
1430                                    1489
1431                                    1490
1432                                    1491
1433                                    1492
1434                                    1493
1435                                    1494
1436                                    1495
1437                                    1496
1438
```

```

333     epsilon_f = self.epsilon_0 / (1 + 370
334         beta * n)    ## DECAYING EPSILON 371
335     Done = 0
336
337         ## SET DONE TO ZERO (BEGINNING
338         OF THE EPISODE)
339
340         i = 1
341
342         ## COUNTER FOR NUMBER OF ACTIONS
343
344         S, X, allowed_a = env.
345             Initialise_game()    ## 375
346             INITIALISE GAME 376
347
348         X = np.expand_dims(X, axis=1)
349             ## MAKE X A
350             TWO DIMENSIONAL ARRAY
351
352         X = np.copy(np.vstack((np.array 378
353             ([[1]]), X))) # add bias term 379
354
355         if self.method == "sarsa":
356             # compute Q values for the given 380
357             state 381
358             a1, h1, a2, Qvalues = self.
359                 forward(X) # -> shape (O,
360                 1)
361
362             # choose an action A using 382
363             epsilon-greedy policy 383
364             A_binary_mask, A_ind =
365                 EpsilonGreedy_Policy(Qvalues, allowed_a, epsilon_f) #
366                 -> shape (O, 1) 385
367
368         while Done==0:
369
370             ##
371             START THE EPISODE
372
373             if (self.method == "qlearning") 386
374             or (self.method == "dqn"): 387
375                 # compute Q values for the 388
376                 given state 389
377                 a1, h1, a2, Qvalues = self.
378                     forward(X) # -> shape 390
379                     O, 1)
380
381                 # choose an action A using 391
382                 epsilon-greedy policy 392
383                 A_binary_mask, A_ind = 393
384                     EpsilonGreedy_Policy(
385                         Qvalues, allowed_a,
386                         epsilon_f) # -> shape 394
387                     O, 1)
388
389                 # take action and observe reward 395
390                 R and state S_prime 396
391                 S_prime, X_prime,
392                     allowed_a_prime, R, Done = 397
393                     env.OneStep(A_ind)
394                 X_prime = np.expand_dims(X_prime 398
395                     , axis=1) 399
396                 X_prime = np.copy(np.vstack((np 400
397                     array([[1]]), X_prime))) 401
398                     add bias term 402
399
400                 n_steps += 1
401
402                 if self.method == "dqn": 403
403                     # store the transition in 404
404                     memory
405                     self.replay_memory.push(X, 405
406                         A_ind, R, X_prime, Done)
407
408             # sample a batch of
409             transitions
410             transactions = self.
411                 replay_memory.sample(
412                     self.batch_size)
413             # turn list of transactions
414             into transaction of
415             lists
416             batch = Transition(*zip(*
417                 transactions))
418
419             # backward step and
420             parameter update
421             self.backward_dqn(batch,
422                 self.gamma)
423
424             # update Q values indirectly by
425             updating the weights and
426             biases directly
427
428             if Done==1: # THE EPISODE HAS
429                 ENDED, UPDATE...BE CAREFUL,
430                 THIS IS THE LAST STEP OF THE
431                 EPISODE
432
433                 if (self.method == "
434                     qlearning") or (self.
435                         method == "sarsa"):
436                     # compute gradients and
437                     update weights
438                     self.backward(R, X,
439                         Qvalues, None, a1,
440                         h1, a2, None,
441                         future_reward=False,
442                         action_binary_mask=
443                             A_binary_mask)
444
445                 # store history
446                 # todo: record max possible
447                 reward per episode
448                 self.R_history[n] = np.copy(
449                     R) # reward per episode
450                 self.N_moves_history[n] = np
451                     .copy(i) # nr moves per
452                     episode
453
454                 # store norm of gradients
455                 self.dL_dW1_norm_history[n]
456                     = np.linalg.norm(self.
457                         dL_dW1)
458                 self.dL_dW2_norm_history[n]
459                     = np.linalg.norm(self.
460                         dL_dW2)
461
462                 # compute exponential moving
463                 average (EMA) to
464                 display during training
465                 ema = alpha*R + (1-alpha)*
466                     ema_previous
467                 if n == 0: # first episode
468                     ema = R
469                 ema_previous = ema
470                 if run_number is not None:
471                     episodes.set_description
472                         (f"Run = {run_number
473                             }; EMA Reward = {ema
474                                 :.2f}")
475                 else:
476                     episodes.set_description
477                         (f"EMA Reward = {ema
478                             :.2f}")
479
480                 break

```



```

407                                     447             self.W1_target = np.copy(
408         else: # IF THE EPISODE IS NOT        self.W1)
              OVER...                    448             self.W2_target = np.copy(
409                                     self.W2)
410         if self.method == "qlearning":
411             # chose next action of policy
412             Q_prime = np.max(self.
                forward(X_prime)
                [-1])
413
414         elif self.method == "sarsa":
415             # chose next action on policy
416             a1_prime, h1_prime,
417                 a2_prime,
                Qvalues_prime = self.
                forward(X_prime)
                -> shape (N_a, 1)
418
419             # chose next action and save it
420             A_binary_mask_prime,
                A_ind_prime =
                EpsilonGreedy_Policy(
                (Qvalues_prime,
                allowed_a_prime,
                epsilon_f)
421
422             # get Qvalue of next action
423             Q_prime = Qvalues_prime[
                A_ind_prime]
424
425         if (self.method == "
426             qlearning") or (self.
                method == "sarsa"):
427             # backpropagation and weight update
428             self.backward(R, X,
                Qvalues, Q_prime, a1
                , h1, a2, self.gamma
                , future_reward=True
                , action_binary_mask
                =A_binary_mask)
429
430         # NEXT STATE AND CO. BECOME
431             ACTUAL STATE...
432         if self.method == "sarsa":
433             A_binary_mask = np.copy(
                A_binary_mask_prime)
434             A_ind = np.copy(
                A_ind_prime)
435             a1 = np.copy(a1_prime)
436             h1 = np.copy(h1_prime)
437             a2 = np.copy(a2_prime)
438             Qvalues = np.copy(
                Qvalues_prime)
439             S = np.copy(S_prime)
440             X = np.copy(X_prime)
441             allowed_a = np.copy(
                allowed_a_prime)
442
443             i += 1 # UPDATE COUNTER FOR
                NUMBER OF ACTIONS
444
445         if (self.method == "dqn") and
            n_steps % self.C == 0):
446             # update target network
                every C steps

```

```

447             self.W1_target = np.copy(
                self.W1)
448             self.W2_target = np.copy(
                self.W2)
449
450         training_end = time.time()
451         self.training_time_in_seconds =
            training_end - training_start
452
453         return None
454
455     except KeyboardInterrupt as e:
456         # return nothing
457         training_end = time.time()
458         self.training_time_in_seconds =
            training_end - training_start
459
460         return None
461
462     def save(self, name_extension=None):
463         # create directory for the model
464         name = f"{self.method}_{self.act_f_1_name}_{
            self.act_f_2_name}"
465         if name_extension is not None:
466             name += f"_{name_extension}"
467
468         path = f"models/{name}"
469         if not os.path.isdir(path): os.mkdir(path)
470         print(f"saving to: {path}")
471
472         # save weights
473         np.save(f"{path}/W1.npy", self.W1)
474         np.save(f"{path}/W2.npy", self.W2)
475
476         # save training history
477         np.save(f"{path}/training_history_R.npy",
            self.R_history)
478         np.save(f"{path}/training_history_N_moves.
            npy", self.N_moves_history)
479         np.save(f"{path}/
            training_history_dL_dW1_norm.npy", self.
            dL_dW1_norm_history)
480         np.save(f"{path}/
            training_history_dL_dW2_norm.npy", self.
            dL_dW2_norm_history)
481
482         # save training parameters and other general
            info
483         params = {
484             "method": self.method,
485             "N_episodes": self.N_episodes,
486             "eta": self.eta,
487             "epsilon_0": self.epsilon_0,
488             "beta": self.beta,
489             "gamma": self.gamma,
490             "alpha": self.alpha,
491             # "gradient_clip": self.gradient_clip,
492             "seed": self.seed,
493             "D": self.D,
494             "K": self.K,
495             "O": self.O,
496             "training_time_in_seconds": self.
                training_time_in_seconds
497         }
498         if self.method == "dqn":
499             params["capacity"] = self.capacity
500             params["batch_size"] = self.batch_size
501             params["C"] = self.C
502             with open(f"{path}/training_parameters.json",
                "w") as f:
503                 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.