

Deep Q-Network to Learn Score Four

Jim Buffat, Aneesh Dahiya, Jonathan Lehner, Till Schnabel

Abstract—In this report, we aim to explore how the complexity of hidden layers affects the learning capacity of an agent in the context of reinforcement learning. We trained a deep neural network to approximate the Q-function of the popular two-player games Connect Four and Score Four. We vary the capacity by changing the number of hidden layers and the corresponding activation function. To evaluate the performance, we picked several test cases and observed how the agent performs. As baseline, we implemented a player with a few hard-coded rules. We compared this player with the trained networks and with a player trained on the same objective, but with linear regression, which served as another baseline. We found that the neural networks outperform the linear regression player compared to the hard-coded player. Our results concerning the network architecture are inconclusive.

I. INTRODUCTION

Ever since AlphaGo beat the world champion for Go, there has been an increased interest in the field of reinforcement learning. The tasks, which seemed hard due to large state- and action-spaces, proved solvable when Mnih et al. approximated the Q-function with a deep neural network.

Although there is extensive research on deep neural networks in the context of computer vision and natural language processing, we observed that deep learning in the context of reinforcement learning from an architectural point of view is not much explored. In this report, we aim at extending this research by varying the complexity and activation of hidden layers of the agent. To quantify our results, we evaluated different agents based on their performance in the games Connect Four and Score Four. We performed the evaluation with the scenarios explained in section II-D. Connect Four and Score Four are perfect information games, i.e. all information conditioning an action is accessible to all players.

A. Connect Four

“Connect Four” [1] is a two-player board game. Traditionally, Connect Four has a 7×6 field to place game tokens, where the vertical (second) dimension can only be filled bottom to top due to gravity constraints. The players take turns in putting a token (different color for each player) anywhere along the horizontal dimension as long as there is empty space. The goal of the game is to form a sequence of four along an arbitrary axis. The first player to form such a sequence wins.

B. Score Four

Similar to Connect Four, “Score Four” [2] is also a two-player board game but with an additional dimension. The

lengths along all dimensions are adjusted to be equal, i.e. it’s a $4 \times 4 \times 4$ field. The gravity constraint is still present along one of the dimensions. As in Connect Four, the goal is to form a sequence of four along an arbitrary axis, the first player to form such a sequence wins. Unlike Connect Four, the player can choose to place the token in 2 dimensions (along the length and the width). Figure 1 shows a few instances of the two games.

C. Deep Q-learning

Before starting, we’ll explain a few terms that are used in the report.

1) *Environment*: The world in which the agent resides. It provides the reward and next state in response to the action of the agent. States change according to some Markov Decision process (MDP).

2) *Agent*: The actor which influences the environment by taking actions.

3) *Policy*: The mapping of state ($x \in X$) action ($a \in A$) pairs which dictates the agent to take an action given the state, i.e. $\pi : X \rightarrow A$. It induces a Markov chain $X_0, X_1, \dots, X_t, \dots$ with transition probabilities

$$P(X_{t+1} = x' | X_t = x) = P(x' | x, \pi(X))$$

4) *Value Function*: The expected reward $r(x, a)$ that an agent can get given a state.

$$V^\pi(x) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t)) | X_0 = x \right],$$

where γ measures the influence of future rewards. Every Value function V induces a greedy policy $\pi_V(x)$ and every policy π induces a value function V^π .

$$\pi_V(x) = \arg \max_{\pi} V^\pi(x) \quad (1)$$

This leads to the Bellman Theorem [3] for the optimal policy V^* :

$$V^*(x) = \max_a [r(x, a) + \gamma \sum_{x'} P(x' | x, a) V^*(x')]$$

There are two approaches to tackling reinforcement learning: the model-free and model-based approach. In the former, we try learning the value function alone. In the latter, we try learning the parameters of the underlying MDP.

Q-learning [4] is a model-free reinforcement learning algorithm which directly estimates the value function.

$$V^*(x) = \max_a Q^*(x, a).$$

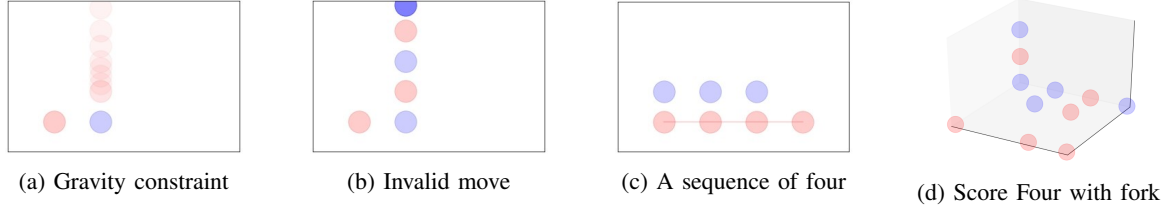


Figure 1: Instances of Connect Four and Score Four (4×4 and $4 \times 4 \times 4$). 1a: Falling token along the vertical axis. 1b: Invalid move 1c: A sequence of 4 leading to win. 1d: Red creates a fork.

where

$$Q^*(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x')$$

Q^* is estimated through samples (state x , action a , reward r , next state x') drawn online from the interaction between environment and agent, as explained below.

$Q(x, a) \leftarrow (1 - \alpha_t) Q(x, a) + \alpha_t (r + \gamma (\max_{a'} Q(x', a')))$
It has been shown that if α_t satisfies the Robbins-Monro condition for stochastic approximation, then the Q-values converge to their optima.

For discrete state-action spaces the function can be modelled as a look-up table (Q-Table) and we can observe that the algorithm has a memory overhead for environments with large state and action spaces. To store each Q-value, we need space of order $\mathcal{O}(|X||A|)$ and it requires time of order $\mathcal{O}(|A|)$ for picking up the best action given a state and updating the Q-value at each iteration.

Deep Q-learning (DQN, by Mnih et al. [5]) proposes a deep neural network that uses a deep neural network to approximate $Q(x, a)$, i.e.

$$Q(x, a) \approx Q(x, a; \theta),$$

where the weights θ encode the network. By using a neural network, one can find a smaller representation of the Q-Table which can be too expensive to store directly as was stated above. It should be noted that for large state-action spaces, even a substantial number of weights yields a high compression rate.

The Q-function is learnt by minimizing the loss between the target Q-value (y) and the Q-value of the network $Q(\cdot; \theta)$. For a transition (x, a, r, x') , the target values are calculated as follows:

$$y = \begin{cases} r & \text{if } x' \text{ is terminal} \\ r + \gamma (\max_{a'} Q(x', a'; \theta)) & \text{else} \end{cases}$$

The usual choice for loss function are squared loss, Huber loss, RMS drop etc. For our networks we are using squared loss.

D. Training Data

The data for training the network was collected by self-play within the environment simulating the games. Since

both the games are multiplayer, the opponent's moves were played by an instance of the same network. The opponent and the main player network are trained simultaneously. We used epsilon greedy policy to explore the large state-action space of the games. For updating the weights transitions are sampled from a memory buffer (experience replay as in Mnih et al.). Additionally we included the improvements to the algorithm proposed by Hasselt et al. called Double DQN. Algorithm 1 illustrates the whole process.

II. MODELS AND METHODS

A. Environment

We defined our own environment inheriting from the base environment class of OpenAI's gym [7]. The environment for Connect Four and Score Four maintained the current state of the board, status of game and current player at all times. Each state-action pair (x, a) had a reward attached to it.

$$r = \begin{cases} 5 & ; x' \Rightarrow \text{current player wins} \\ -10 & ; x' \text{ lets the next player win} \\ 0 & ; x' \text{ The game ends in tie} \\ -100 & ; \text{Player takes an invalid action} \\ 0 & ; \text{otherwise} \end{cases}$$

Here x' denotes the next state. We observed that awarding the losing reward before the opponent makes a winning move leads to better results in the sense that the agent preemptively learns to block the winning moves of the opponent. A game is finished when a player wins, loses, makes an invalid move or there is no more space left on the board (tie).

B. Network Architecture

We used the Keras-RL [8] higher level API with tensorflow-backend [9] for the network design and training. Keras-RL doesn't have support for multi-agent networks, but it could be made into one by overloading the forward and backward pass functions for the the Agent class from Keras-RL. In this project, we used Margus Niitsoo's [10] implementation for the multi-agent network.

Algorithm 1: Double Deep Q-learning with experience replay [5] and [6]

Data: γ , Network architecture \mathcal{Q}_{NN} , empty state x_0

Result: Trained agents $\mathcal{Q}_{NN}(\cdot, \cdot; \theta_i^t)$ with $i \in \{1, 2\}$

Initialize: Replay memory capacity;

Network weights θ ;

Copy θ for both players and both policy and target nets, i.e. $\theta_1^p \leftarrow \theta$, $\theta_2^p \leftarrow \theta$, $\theta_1^t \leftarrow \theta$, $\theta_2^t \leftarrow \theta$;

for $episode = 1, M$ **do**

 Initialize the starting state $x \leftarrow x_0$

for each time step t **do**

1. Set player $i \leftarrow t \bmod 2$, $\theta^p \leftarrow \theta_i^p$,
 $\theta^t \leftarrow \theta_i^t$
- // Play game.
2. Select action $a^* = \arg \max_a \mathcal{Q}_{NN}(x, a; \theta)$
 or random with prob ϵ
3. Execute selected action a in environment
4. Observe next state x' and reward r
5. Store experience (x, a, x', r) in replay
 memory and update state $x \leftarrow x'$
- // Update weights
6. Sample random batch (x_b, a_b, x'_b, r_b) from
 replay memory
7. Calculate the expected Q-value with the
 target net, i.e.
 $y_b = \gamma * \mathcal{Q}_{NN}(x_b, a_b; \theta^t) + r_b$. (= reward
 for terminal states and discounted reward
 based on current policy network for non
 terminal states)
8. Calculate the Q-value from the policy
 network, i.e. $q_b = \mathcal{Q}_{NN}(x_b, a_b; \theta^p)$
9. Calculate loss $\mathcal{L}(y_b, q_b)$
10. Make an update step to the weights of
 the policy net θ^p to minimize the loss by
 gradient descent

if $t == T$ **then**

$\theta^t \leftarrow \theta^p$

end

end

1) *Input and output layer:* Input and output layer were common to all the networks used. The input layer consisted of a flattening unit which changed the shape of input state to 1 dimension. Activations followed each layer (the input one was always followed by tanh) except the output layer. In the output layer, the number of units were equal to the size of action space, i.e. each unit represented a Q-value for the corresponding action in action space.

2) *Hidden layers:* One can increase the complexity of a neural network either by increasing the number of layers or increasing the number of units. We chose the former.

Each hidden layer had 50 units. We increased the number of hidden layers from one to four in steps of one.

3) *Activation:* We evaluated both ReLU and tanh activation for our networks. For each variation in number of layers there was a corresponding variation in activation.

agent_	Description
5_relu	1 hidden layer with 50 units Activation: ReLU
5:5_relu	2 hidden layers with 50 units each Activation: ReLU
5:5:5_relu	3 hidden layers with 50 units Activation: ReLU
5:5:5:5_relu	4 hidden layers with 50 units Activation: ReLU
5_tanh	1 hidden layer with 50 units Activation: Tanh
5:5_tanh	2 hidden layers with 50 units each Activation: Tanh
5:5:5_tanh	3 hidden layers with 50 units Activation: Tanh
5:5:5:5_tanh	4 hidden layers with 50 units Activation: Tanh
All the agents have the same input and output layer. as described in section II-B1	

Table I: Description of the policy network for different agents.

To sum up, for each game we trained 8 networks. Figure 2 gives an overview of the architecture. Table I concisely describes the different architecture choices.

C. Training

We trained each network for 500 000 steps with the Adam optimizer [11]. Table II summarizes the training parameters used for all the networks. These were fixed, so that the networks had comparable training conditions. Since our goal was to check which network architecture performs the best given the conditions, fixing these parameters was necessary.

Reward discount factor γ	0.99
ϵ -Greedy Policy	0.5
Experience replay length	50000
Mini-Batch size	32
Warm up steps	100
Loss function	Squared Loss
Adam optimizer's parameters	
Learning rate	0.001
β_1	0.9
β_2	0.999

Table II: Training Parameters

D. Agent Evaluation

We tested our agents against two types of players: AlmostRandomPlayer (ARP) and LinearRegressionPlayer (LRP) described below. We use these two players as baseline to compare our models.

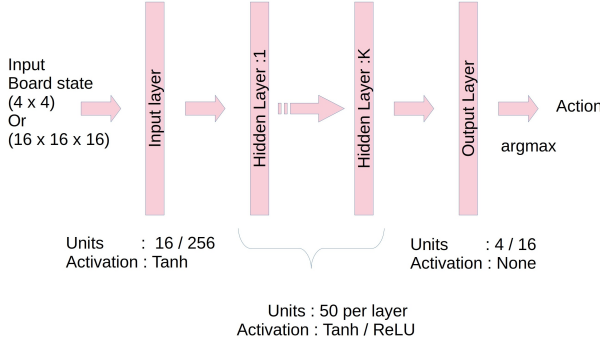


Figure 2: Network Architecture: The input layer flattens the board to 1D. We evaluated 1-4 hidden layers. The output layer gives the Q-value for each action of which the maximum is chosen as final action.

1) *LRP*: This player is similar to the agents described in section II-B. The only difference is it doesn't use any non-linearity as an activation function. It approximates the Q-value by linear regression. This agent was trained using the same parameters as described in section II-C. It wasn't directly pitted against the agents mentioned in Table I, rather its performance against ARP was used as a baseline.

2) *ARP*: This player's policy is outlined in Algorithm 2. It anticipates a direct winning move for itself and one for his opponent, i.e. the ARP will put a fourth token to a row of three. If not possible, it makes a move which stops the opponent's immediate victory in the next turn. If there's no such move either, it randomly chooses among those moves that don't help the opponent to win. If this is not possible, it randomly chooses among all valid moves.

A skilled human player can beat the ARP in Score Four in six moves with a so-called "fork". This describes a situation where the player has two or more rows with three tokens, s.t. the opponent cannot block all of them (see again Figure 1d). As long as the ARP is not lucky with its random guesses, it won't see this fork coming before it's too late.

3) *Specific Scenarios*: We also performed a few checks on the agents in the following scenarios.

- 1) Making a valid move: During our ARP tests, we counted the times the agent made a valid move.
- 2) Making a winning move: We initialized the board to different states from which the agent can win in one move and counted in how many cases it uses that chance.
- 3) Blocking the opponent: We tested how often the agent blocks the opponent from making a direct winning move.
- 4) Making a fork move: We observed how many times the agent uses its chance to create a fork.
- 5) Blocking a fork move: We checked how often the agent blocks the opponent from making a fork move.

Algorithm 2: AlmostRandomPlayer as baseline

Input: Current board state, playerId
Output: Action
Initialize empty RandomActions list;
Initialize ValidActions1 to possible actions the player can take;
for *action* in ValidActions1 **do**
 Take action with playerId;
 if *win* **then**
 return action;
 end
Initialize ValidActions2 to possible actions the opponent can take;
 if *action* in ValidActions2 **then**
 Take action with opponent's player ID;
 if *Opponent not win* **then**
 append action to RandomActions;
 end
 end
end
for *action* in ValidActions1 **do**
 Take action with opponent's player ID;
 if *opponent wins* **then**
 return move;
 end
end
if *length of RandomActions not zero* **then**
 return random action from RandomActions;
end
return random action from ValidActions1;

Of course, we cannot test all possible cases for the described scenarios. For each initialized board state, we had to assure its validity, i.e. that it is a state reachable from two players taking alternating turns. We sampled a few easy test cases of which we had to test each for validity.

III. RESULTS

A. Connect Four

1) *LRP performance*: Table III summarizes the results for the LRP validation as described above.

LRP	starts first			starts second		
	Win	Loss	Invalid	Win	Loss	Invalid
LRP	0.0	4.2	95.8	0.0	18.9	81.1

Table III: Performance of LRP against ARP in Connect Four (in [%]).

2) *Performance with respect to ARP*: Like the LRP, all the agents described in section II-B were pitted against the ARP. The results are shown in Table IV.

Agent starts	first			second		
agent_	Win	Loss	Tie	Win	Loss	Tie
5_relu	13	19	68	6.8	34.6	58.6
5:5_relu	3	9.5	87.5	9.4	18.1	72.5
5:5:5_relu	2.7	22.1	75.2	18.2	26.4	55.4
5:5:5:5_relu	23.9	16	60.1	19	21.1	59.9
5_tanh	7.3	2.71	65.6	21.1	35.2	43.7
5:5_tanh	2.2	26.5	71.3	20.1	49.9	30
5:5:5_tanh	10.5	19.8	69.7	16.4	46.3	37.3
5:5:5:5_tanh	7.5	37.2	55.3	14.6	53.6	31.8

Table IV: Performance with ARP in Connect Four (in [%]).

agent_	Win	Blocking Loss	Fork win	Blocking Fork
5_relu	68	42	33	50
5:5_relu	68	58	50	50
5:5:5_relu	75	42	68	50
5:5:5:5_relu	75	50	50	33
5_tanh	92	25	17	83
5:5_tanh	42	17	33	50
5:5:5_tanh	50	33	17	50
5:5:5:5_tanh	58	25	17	33

Table V: Test how many correct moves the agents take for different specific scenarios in Connect Four (in [%]).

3) *Specific scenario test*: Table V shows how well the different first agents performed in the described scenarios. We created 12 win and loss scenarios each, and 6 fork win and fork loss scenarios each.

B. Score Four

1) *LRP performance*: We used the same setting as in section III-A1. The results are shown in Table VI.

2) *Performance with respect to ARP*: We used the same setting as in section III-A2. Table VII summarizes the results.

3) *Specific scenario test*: Similar to that of Connect Four, table VIII shows how well the different agents performed in the described scenarios. We created 56 win and loss scenarios each, and 20 fork win and fork loss scenarios each.

IV. DISCUSSION

In the Connect Four game, our agents learned to make valid moves as evident from the experiment with ARP (Table IV). The agents performed better than their linear counterpart (LRP), which couldn't learn to make valid moves and didn't win a game against the ARP. Agent 5:5:5:5_relu showed fairly moderate performance where it had roughly the same number of wins and losses. Although the majority of the games did end in tie for this agent as well.

LRP starts	first			second		
Agent	Win	Loss	Invalid	Win	Loss	Invalid
LRP	0.0	7.9	92.1	0.0	8.7	91.3

Table VI: Performance of LRP with ARP in Score Four (in [%]).

Agent starts	first			second		
agent_	Win	Loss	Invalid	Win	Loss	Invalid
5_relu	0.4	99.5	0.1	0.4	99.6	0.0
5:5_relu	1.3	97.7	1.0	0.2	99.0	0.8
5:5:5_relu	2.4	87.8	9.8	0.0	78.8	21.2
5:5:5:5_relu	0.0	6.8	93.2	0.0	29.6	70.4
5_tanh	0.1	99.6	0.3	0.6	99.2	0.2
5:5_tanh	0.6	94.3	5.1	0.4	98.2	1.4
5:5:5_tanh	0.2	92.0	7.8	0.0	97.2	2.8
5:5:5:5_tanh	0.1	91.0	8.9	0.0	93.4	6.6

Table VII: Performance with ARP in Score Four (in [%]).

agent_	Win	Blocking Loss	Fork win	Blocking Fork
5_relu	4	5	15	15
5:5_relu	4	2	10	5
5:5:5_relu	7	7	10	10
5:5:5:5_relu	2	4	15	5
5_tanh	5	5	5	10
5:5_tanh	4	7	5	0
5:5:5_tanh	14	5	5	10
5:5:5:5_tanh	9	7	20	5

Table VIII: Test how many correct moves the agents take for different specific scenarios in Score Four (in [%]).

The performance in view of scenarios described in section II-D reveals that agents did pick obvious winning moves, countering opponent's winning move and countering losing moves for some of the cases (Table V). Despite the small action space ($= 4$) and unremarkable performance against the ARP, the results still qualify the claim for agents' somewhat good performance.

In the Score Four game, our agents performed worse than their performance in Connect Four. From Table VII one can observe the agents didn't learn to make valid moves consistently. Although on comparing this performance with that of LRP vs ARP (Table VI), we could safely say the agents learned to make valid moves better than the LRP. Making arguments or drawing conclusions about performance in specific scenarios isn't justified as the agents didn't learn valid moves for Score Four. Needless to say a skilled human opponent would have no problem beating any of the agents.

V. SUMMARY

We evaluated different neural networks to train our agent via Q-learning on the games Connect Four and Score Four. While some of the trained agents reached the playing level of our ARP with hard-coded rules on Connect Four, none were able to compete with the ARP on Score Four due to its higher complexity. Comments on the performance amongst different agents are qualified by the absence of correlation between the network hyperparameters and overall performance against the ARP. However, one can safely conclude that neural networks are better than linear regression at approximating the Q-function for the two games.

REFERENCES

- [1] Wikipedia contributors, “Connect four — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 12-December-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Connect_Four&oldid=929134351
- [2] —, “Score four — Wikipedia, the free encyclopedia,” 2017, [Online; accessed 12-December-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Score_Four&oldid=818002779
- [3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., ser. Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice Hall, 2010. [Online]. Available: <http://aima.cs.berkeley.edu/>
- [4] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992. [Online]. Available: <https://doi.org/10.1007/BF00992698>
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [6] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: <http://arxiv.org/abs/1509.06461>
- [7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: <http://arxiv.org/abs/1606.01540>
- [8] F. Chollet *et al.*, “Keras,” <https://keras.io>, 2015.
- [9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- [10] M. Niitsoo, “Interleaved agent,” <https://github.com/velochy/rl-bargaining>, 2019.
- [11] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, 12 2014.

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

DEEP Q-NETWORK TO LEARN SCORE FOUR.

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

BUFFAT

DAHIYA

LEHNER

SCHNABEL

First name(s):

JIM

ANEESH

JONATHAN

TILL

With my signature I confirm that

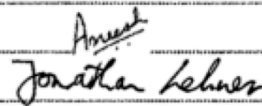
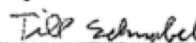

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zuerich, 5 January 2020

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.