university of groningen / faculty of mathematics and natural sciences / artificial intelligence

ADVERSARIAL REINFORCEMENT LEARNING IN A CYBER SECURITY SIMULATION

Bachelor's Project Thesis

Richard Elderman, s2592444, r.elderman@student.rug.nl,
Leon Pater, s2380153, l.j.j.pater@student.rug.nl,
Albert Thie, s1652184, a.s.thie@student.rug.nl,
Supervisors: Dr. M.A. Wiering, Dr. M.M. Drugan

Abstract: In this thesis we focus on the problem of securing assets and data in a computer network. After building our own security game, we train attacking and defending agents against each other. The security game is modelled as a sequential decision making problem for the attacker and defender. The game is simulated as an extensive form game with incomplete information and stochastic elements. By using neural networks, Monte Carlo learning and Q-learning we pit various reinforcement learning techniques against each other to examine their effectiveness against learning opponents. We found Q-learning and Monte Carlo learning with *epsilon*-greedy exploration to be most effective in performing the defender role and different attacker algorithms based on Monte Carlo learning to be most effective in attacking.

1 Introduction

In an ever changing world providing security and safety to individuals and institutions is a complex task. Not only is there a wide diversity of threats and possible attackers, many avenues of attack exist in any target. Targets can range from a website to be hacked to a stadium to be attacked during a major event. We hope to demonstrate the complexity of simulating these wide ranging environments and provide a basis for adaptive learning techniques able to deal with learning opponents.
Whitehead and Ballard (1990) note that an important part of reinforcement learning is adapting to a changing environment by reevaluating the value of the states in the learning algorithm. Lopes, Lang, Toussaint, and Oudeyer (2012) explain the advantage of pairing reinforcement learning with exploration techniques, while Schmidhuber (1991) points out the problem of local minima that such learning algorithms encounter. Lastly the complexity of using traditional machine learning algorithms greatly increases when pitted against each other in an adversarial setting (Huang, Joseph, Nelson, Rubinstein, and Tygar (2011)). With our research, we aim to evaluate the effectiveness of standard reinforcement learning techniques in a newly developed cyber security task which contains stochastic elements.
Contributions To this end we have selected a number of reinforcement learning algorithms (Russell and Norvig (2003); Sutton and Barto (1998); Auer, Cesa-Bianchi, and Fischer (2002); Garivier and Moulines (2008); Wang, Li, and Lin (2013); Wiering and van Otterlo (2012)) and pitted them against each other as a defender and attacker that play against each other in a simulation of an adversarial game with partially observable states. The simulation is modelled as an extensive form game with incomplete information (Myerson (1991); Roy, Ellis, Shiva, Dasgupta, Shandilya, and Wu (2010)). The simulation is played on a network consisting of nodes representing a server or network connected with each other. The nodes are modelled based on the ten most used attack types in on line hacking, as defined by the 2013 review of the OWASP organisation (OWA).
The attacker must attempt to find a way through a network consisting of various locations, to reach and penetrate the location containing the important asset. The defender can prevent this by either protecting certain avenues of attack by raising its defense or choosing to improve the capability of the location to detect an attack in progress. The detection of an attack is an important aspect. Sharma, Kalbarczyk, Barlow, and Iyer (2011) showed that

62% of the incidents in the study were detected only after attacks have already damaged the system. Recognizing attacks is therefore crucial for improving the system. The balance between prevention and detection is a delicate one, which brings unique hurdles with it.
This creates an environment where the attacker cannot simply execute previously successful strategies, but must adapt to the defender's strategies as well. The environment resembles a dynamic maze that is constantly changing at the end of a game due to the opponent's behavior. As the attacker becomes more successful in successive games, the defender will adapt, creating new situations where different paths must be chosen by the attacker. The same holds true the other way around. This process becomes more complicated by the fact that in the beginning the attacker has no knowledge of the network. The attacker can only gain access to knowledge about a part of the network by penetrating the defenses of that part. The defender in turn does know the internal network, but not the position or type of attack of the attacker. Adapting different strategies to unobservable opponents is crucial for dealing with the realities of cyber attacks. (Chung, Kamhoua, Kwiat, Kalbarczyk, and Iyer (2016)). Attackers often exploit previously unknown vulnerabilities, attack unknown locations in a network at unknown times and often use secondary software to obfuscate their movements (OWA). This makes it essential for any defending technique to be able to deal with a changing and partially observable environment.
Reinforcement learning techniques for the attacker that are used are Monte Carlo learning, with some exploration strategies: ϵ-greedy, Softmax, Upper Confidence Bound 1 (Auer et al. (2002)) and Discounted Upper Confidence Bound (Garivier and Moulines (2008)), and backward Q-learning with ϵ-greedy exploration (Wang et al. (2013)). The defender uses the same algorithms and two different neural networks with back-propagation as extra algorithms.
Outline In Section 2 the cyber security game is described and explained. In Section 3 the used reinforcement learning techniques and algorithms are described. Section 4 explains the experimental setup, and Section 5 presents the experimental results for the simulations with learning agents and discusses these results. Finally, in Section 6 the main findings are summarized and future work is proposed. Section 7 shortly explains the individual contributions.

2 Cyber Security Game

In cyber security in the real world, one server with valuable data gets attacked by many hackers at the same time, depending on the value of the data content. It is also often the case that one network contains more than one location with valuable data. However, in the simulation made for this study, only one attacker and one defender play against each other, while there is only one asset. This is chosen for sake of little complexity of the "world", such that agents do not have to learn very long before they become aware of any good strategy. A bigger world would lead to longer games and a bigger state space, hence it would be more difficult for each agent to find any optimal policy. This would also slow down the adversarial learning effects.

2.1 Network

In the simulation the attacker and defender play on a network representing a part of the internet. The network consists of nodes, which are higher abstractions of servers in a cyber network, such as login servers, data servers, internet servers, etc. Nodes can be connected with each other, which represents a digital path: it is possible to go from one server to another one, only if they are (in)directly connected. Every connection is symmetric, so if nodes A and B are connected, it is possible to go from A to B and from B to A.
In a network there are three types of nodes:

1. The start node, which is the node in which the attacker is at the beginning of each game. It has no asset value. It can be seen as the attacker's personal computer.

2. Intermediate nodes. These are nodes without asset values, in between the start node and the end node. They must be hacked by the attacker in order to reach the end node.

3. The end node, which is the node in the network containing the asset. If the attacker successfully attacks this node, the game is over and the attacker has won the game.

Each node consists of 10 different attack values, 10 different defense values, and a detection value. Each value in a node has a maximal value of 10. In the standard network of 4 nodes, this creates a possible 10^{84} environmental states. The attack and defense values are paired, each pair represents the attack strength and security level of a particular hacking strategy. The detection value represents the strength of detection, which represents the chance that, after a successfully blocked attack, the hacker can be detected and caught. The environmental state of the network can be summarized as a combination of the attack values, defense values and detection value of each node. Both the attacker and defender agents have internal states representing parts or the entire environmental state, on which they base their actions.
The 10 different hacking strategies are the top 10 most critical web application security risks, according to the OWASP top 10 from 2013 (OWA), respectively: an injection, broken authentication and session management, cross-site scripting(XSS), insecure direct object references, security misconfiguration, sensitive data exposure, missing function level access control, cross-site request forgery (CSRF), using known vulnerable components and unvalidated redirects and forwards.
The game is a stochastic game due to the detection-chance variable. When an attack fails (which is extremely likely), a chance equal to the detection parameter determines if the attacker gets caught. This resembles a real-life scenario because attacks can remain unnoticed, and the chance that an attack remains unnoticed decreases when there is more detection.

2.2 Agents

In the simulation there are two agents, one defender agent, and one attacker agent. Each agent has limited access to the network, just like in the real world, and it has only influence on its side of the values in the nodes (security levels or attack strengths). Both agents have the goal to win as many games as possible. Reinforcement learning techniques will help them doing this.

2.2.1 Attacker Agent

The attacker has the goal to hack the network and get the asset in the network. He can do so by successfully attacking the node in the network in which the asset is stored.
For every node that is accessible by the attacker, only the values for the attack strengths are known by the attacker. The attacker has access to the node he currently is in, and he knows which nodes are accessible from that server. He can only attack those nodes directly attainable from the current node, and he can do so by incrementing attack values on those nodes. Per game step he is allowed to only increment one attack value on one of the accessible nodes, that has not already the maximal value. The attacker agent represents this information as separate states, containing the starting node, the node to be attacked, the type of attack and the current strength of the attack type. These partial attack states can be mapped on specific environmental states. The action to change an attack value to 4 of node 2 and attack type 7 for example can only occur if in the current environmental state attack type 7 in node 2 has an attack value of 3. By executing this move the attacker changes the environmental state to set the attack value of attack type 7 in node 2 to 4.

2.2.2 Defender Agent

The defender has the goal to detect the attack agent before he has taken the asset in the network. He can do so by incrementing security and detection levels on the nodes.
The defender has, as in the real world, access to every node in the network (recall that the network only represents a part of the internet). However, the defender only knows the defend and detection values on each node: he does not know the attack values.
Per game step, the defender is allowed to increment one defend or detection value on one node in the network. By incrementing a defend value, it becomes harder for the attacker to hack the node using that attack type, but this might not be needed because the defense is already strong enough to stop an attack of that type. By incrementing the detection value, the chance that an unsuccessful attack is detected becomes higher, but

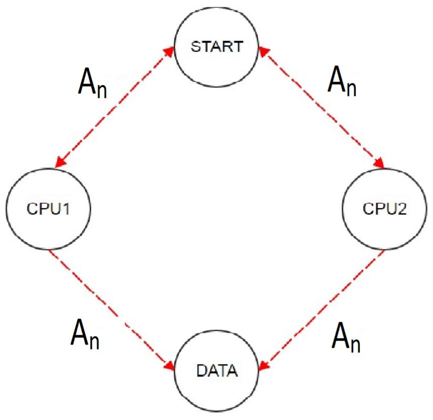

Figure 2.1: Attacker perspective. Arrows indicate possible attacks A with attack type n from the current node. All initial attack values are equal to zero. The attacker starts at the start-node.

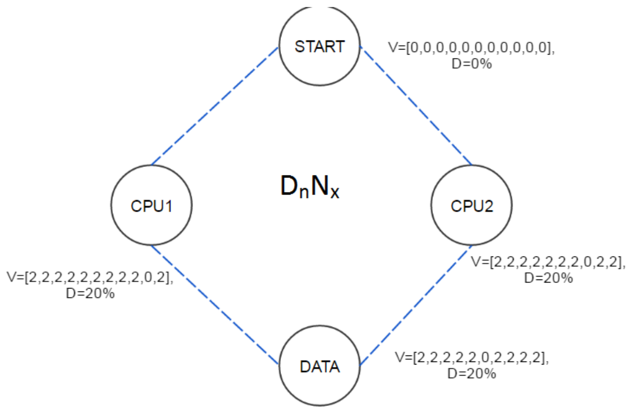
START
V=[0,0,0,0,0,0,0,0,0,0,0], D=0%
D_nN_x
CPU1
CPU2
V=[2,2,2,2,2,2,2,2,2,0,2], D=20%
V=[2,2,2,2,2,2,2,0,2,2], D=20%
DATA
V=[2,2,2,2,2,0,2,2,2,2], D=20%

Figure 2.2: Defender perspective. The V values indicate the initial defense values. D stands for the detection-chance if an attack fails. The defender has full access to the network and has possible defend actions D on defense type n and detect actions, on node N

when the defense is not strong enough a high detection chance is completely useless. The Monte-Carlo and Q-learning agents do not have an internal state representation, but base their actions on previous success, regardless of the environmental state. The neural and linear networks use the entire environmental state as an input. Just like the attacker agents the defender agents modify the environmental state of the network by performing a move.

2.3 Standard Network

In the simulations the agents play on the standard network, consisting of four nodes connected with each other in a diamond shape. The start node is connected with two intermediate nodes, which both are connected with the end node (see Figures 2.1 and 2.2).

All attack and defense values on the start node are initially zero: both agents need to learn that putting effort in this node has no positive effect on their win rate, since the attacker cannot take the asset here and the defender therefore does not need to defend this node. Both intermediate nodes have a major security flaw represented by one attack type having an initial security level of zero, leaving the attacker two best actions from the start node. The node containing data has a similar security flaw, which gives the attacker a clear path towards its goal. The defender must identify this path and fix the security flaws by investing in the security level of the attack type with initially low security on the node, and in turn the attacker has to find another path. If both agents learn well, the attacker can only win by luck, since the defender has time to invest in the security of the attack type with an initially low security level in the end node. Figures 2.1 and 2.2 recap the game from respectively the attacker and the defender perspective.

2.4 Game Procedure

At the start of each game, the attacker is in the start node. The values in all the nodes are set to their initial values. Then one step in the game is made. Both agents choose an action from their set of possible actions. The actions will be performed in the network, and the outcome is determined. In every game step, only the incremented attack value is evaluated. So on the node on which the attack value of an attack type was incremented, it is determined if the attack was successful. This is only the case if the attack value for the attack type is higher than the defend value for that attack type: only if the attack overpowers the defense ($attackvalue = defensevalue + 1$), the attack was successful. In that case the attacker moves to the

Table 2.1: An example game showing the attacker moving to CPU2 and subsequently to the DATA node. The defender performs two defense actions in CPU1

Gamestep	Action Attacker	Action Defender	Node	Attack Values	Defense Values
0	-	-	START	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
			CPU1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 2, 2, 2, 2, 2, 2, 2, 2, 2]
			CPU2	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 0, 2, 2, 2, 2, 2, 2, 2, 2]
			DATA	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 2, 0, 2, 2, 2, 2, 2, 2, 2]
1	CPU2, Attack Type 2	CPU1, Defense Type 1	START	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
			CPU1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 2, 2, 2, 2, 2, 2, 2, 2, 2]
			CPU2	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 0, 2, 2, 2, 2, 2, 2, 2, 2]
			DATA	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 2, 0, 2, 2, 2, 2, 2, 2, 2]
2	DATA, Attack Type 3	CPU1, Defense Type 1	START	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
			CPU1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
			CPU2	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]	[2, 0, 2, 2, 2, 2, 2, 2, 2, 2]
			DATA	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	[2, 2, 0, 2, 2, 2, 2, 2, 2, 2]

attacked node. If this node is the end node, the game ends with a win for the attacker. If the attack value of the attack type is lower than or equal to the defense value, the attack was blocked. In that case, it is determined if the attack was detected. This is done using the detection value of the node that was attacked. That value is a number between 0 and 10, the chance to be detected is (detection value * 10)%. So for the detection value being 4, there is 40% chance that the attack was detected (and for detection value=10, every blocked attack will always be detected). If the attack was indeed detected, the game ends and the defender wins. If the attack was not detected, another game step is played. Note that, because no value can be higher than 10, the maximal number of steps in one game is 10 attack types*10 values*the number of nodes. In practice the games do almost never reach this number of steps, but when it does the simulation must be restarted (draws are not possible).
At the end of each game, the winner gets a reward with the value 100, and the loser gets a reward with the value -100. One agent's gain is equivalent to another's loss, and therefore this game is a zero-sum game (Neumann and Morgenstern (2007)).

2.4.1 Example Game

An example game on the standard network will be: in game step 1 the attacker attacks the gap in one of the intermediate nodes, while the defender fixes the flaw in the other node. The attacker now moves to the attacked intermediate node. Then the attacker attacks the security flaw in the end node, while the defender increments the same value as in the previous game step. Now the attacker moves to the end node and has won the game. The agents update the Q-values of the state-action pairs played in the game, and a new game will be started.
A game on the standard network lasts at least 1 time step, and at most 400 time steps. Table 2.1 shows a game where the attacker wins in two time steps.

3 Reinforcement Learning

The agents in the simulation need to learn to optimize their behavior, such that they win as many games as possible. In each game step, an agent needs to choose an action out of a set of possible actions. We can model this as a multi-armed bandit problem (MAB, Mahajan and Teneketzis (2008)) by each arm representing one of the attack types on one of the nodes. Agents base their decision on the current state of the network. States are defined differently for both agents, because the complete state of the network is partially observable for both agents, and they have access to different kinds of information about the network. The agents also have different actions to choose from. In each state the set of possible actions is different, and there are many possible states for both agents.
An attacker has access to the node he currently is in and the nodes directly connected to that node, and only to the attack values on those nodes. States are for an attacker agent defined as $s(n)$, where n is the node the agent is currently in. A node is for the attacker defined as $n(a_1, a_2, ..., a_{10})$, where each a is the attack value of an attack type on the node. An

action for an attacker is defined as $A(n', a)$, where n' is one of the neighbouring nodes of node n that is chosen to be attacked and $a \in a_1, a_2, ..., a_{10}$, where each a is a different attack type with respect to the OWASP top 10 (OWA). In each state, the number of possible actions is 10 * the number of neighbours of n minus the actions that increment any attack value that is already maximum (has the value 10), since 10 is the predefined maximum for the abstract investment values.
The defender has access to the entire network, but only to the defense and detection values on the nodes. For a defender agent a state is defined as $s(n_1, n_2, ..., n_i)$, where each n is a different node in the network. A node is for the defender defined as $n(d_1, d_2, ..., d_{10}, det)$ where each d is the defense value of an attack type on the node and det is the detection value on the node. An action for a defender is defined as $A(n, a)$, where n is the node in the network that is chosen to invest in, and $a \in d_1, d_2, ..., d_{10}, det$, where each d is a different attack type, and det is the detection parameter. In each state, the number of possible actions is 11 * the number of nodes minus the actions that increment the defense value of an attack type that is already maximum (has the value 10).
Seven different learning algorithms are implemented to optimize the agent's behavior: four types of Monte Carlo learning, each with a different exploration algorithm, Q-learning with ϵ-greedy exploration, and two neural networks one with and one without a hidden layer. The neural networks are only implemented for the defender agent.

3.1 Monte Carlo Learning

In the first reinforcement learning technique, agents learn using Monte Carlo learning (Sutton and Barto (1998); Wiering and van Otterlo (2012)). The agents have a table with all possible state-action pairs, along with estimated reward values.
After each game the agent updates the estimated reward values of the state-action pairs that were selected during the game. In contrast to Q-Learning, that takes potential future state values into account, Monte Carlo learning updates each state the agent visited with the same reward value, using this formula:

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha * (R - Q_t(s, a))$$

Learning rate α is the learning rate which is a parameter ranging from 0 to 1 that represents how much the agent should learn from a new observation. A value of one indicates that the agent fully counts the new observation, where a value of zero does not consider any new information. α can be changed to optimize the learning effect for an agent.
Reward R is the reward obtained at the end of each game.
Estimated reward The Q-values are the estimated reward values. They represent how much the agent expects to get after performing an action. The s is the current state of the world, for the attacker the node he currently is in and for the defender it is empty: the state s has always the value 0. The a is a possible action to do in that state (see also the start of this section). and for the defender it is empty: the state s has always the value 0.

3.1.1 Exploration Strategies

In reinforcement learning it is always a dilemma between choosing the action that is considered best (exploitation) and choosing some other action, to see if that action is better (exploration) (Thrun (1992); Wiering and Schmidhuber (1998)). It is always the goal to balance these two options out such that the regret, the loss because of making bad decisions, is minimized (Berry and Fristedt (1985)).
For Monte Carlo learning, four different exploration algorithms are implemented that try to deal with this problem in the cyber security game. The four algorithms are ϵ-greedy, Softmax, Upper Confidence Bound 1 (Auer et al. (2002)), and Discounted Upper Confidence Bound (Garivier and Moulines (2008)), which we will now shortly describe.

3.1.2 ϵ-greedy strategy

In the standard model, the ϵ-greedy exploration strategy is implemented. This strategy selects the best action with probability $1 - \epsilon$, and in the other cases it selects a random action out of the set of possible actions.
Exploration ϵ is here a value between 0 and 1, determining the amount of exploration (Russell and Norvig (2003)). The higher this value, the more there will be explored.

3.1.3 Softmax

The second exploration strategy is Softmax. This strategy gives every action in the set of possible actions a chance to be chosen, based on the estimated reward value of the action. Actions with higher values will have a bigger chance to be chosen (Sutton and Barto (1998); Wiering and van Otterlo (2012)). A Boltzmann distribution is used in this algorithm to calculate the chances:

$$P_t(a) = \frac{e^{\frac{Q_t(s,a)}{\tau}}}{\sum_{i=1}^{n} e^{\frac{Q_t(s,i)}{\tau}}}$$

Selection chance $P_t(a)$ is the chance that action a will be chosen.
Estimated reward $Q_t(s,a)$ is the estimated reward value for action a in state s at time step t.
Set of possible actions n is the total number of possible actions.
Temperature τ is the temperature parameter, which indicates the amount of exploration. The higher the temperature, the more there will be explored.

3.1.4 UCB-1

The third exploration strategy is called Upper Confidence Bound 1 (UCB-1) (Auer et al. (2002)). This algorithm bases its action selection on the estimated reward values and on the number of previous tries of the action. The less an action is tried before, the lower the confidence about the value and the higher the exploration bonus that is added to that value for the action selection. At the start of the simulation, actions will be chosen that have not been tried before. After no such actions are left, the action will be chosen that maximizes $V_t(a)$, computed by:

$$V_t(a) = Q_t(s,a) + \sqrt{\frac{c * \ln n}{n(a)}}$$

Estimated reward value $Q_t(s,a)$ is the estimated reward value for action a in state s at time step t.
Previous tries $n(a)$ is the number of previous tries of action a over all previously played games in the simulation.
Total previous tries n is the total number of previous tries for all actions currently available over all previously played games in the simulation.
Confidence rate c is the confidence rate, which indicates the rate of confidence of the estimated reward values. The higher the confidence rate, the more there will be explored.

3.1.5 Discounted UCB

The last exploration strategy is called Discounted Upper Confidence Bound (Discounted UCB), and is a modification of the UCB-1 algorithm (Garivier and Moulines (2008)). It is based on the same idea, but more recent previous tries have more influence on the value than tries longer ago. This algorithm is an improvement of the UCB-1 algorithm when used in a non-stationary environment like the simulation in this study.
Like in UCB-1, at the start of the simulation actions will be chosen that have not been tried before, and after no such actions are left the action will be chosen that maximizes $V_t(a)$, computed by:

$$V_t(a) = Q_t(s,a) + 2B\sqrt{\frac{\xi \log n_t(\gamma)}{N_t(\gamma,a)}}$$

Estimated reward $Q_t(s,a)$ is the estimated reward value for action a in time step t.
Maximal reward B is the maximal reward that can be obtained from a game.
Confidence rate ξ is the confidence rate, which indicates the rate of confidence of the estimated reward values. It has the same role as the c-parameter in the ucb-1 exploration strategy.
Global sum $n_t(\gamma)$ is defined as:

$$n_t(\gamma) = \sum_{i=1}^{K} N_t(\gamma,i)$$

Possible actions K is the set of possible actions in time step t.
Local sum $N_t(\gamma,i)$ is defined as:

$$N_t(\gamma,i) = \sum_{s=1}^{t} \gamma^{t-s} \mathbb{1}_{\{a_s=i\}}$$

Selection condition $\mathbb{1}_{\{a_s=i\}}$ is the condition that the value for time step s must only be added to the sum if action i was performed in time step s.
Discount factor γ is the discount factor that determines the influence of previous tries on the estimated reward values.

3.2 Q-Learning

Q-learning is a model-free reinforcement learning technique. In Watkins and Dayan (1992) it was proved that Q-learning converges to an optimal policy for a finite set of states and actions for a single agent.
The Backward Q-learning algorithm (Wang et al. (2013)) is used to solve the multi-agent sequential decision making problem. Preliminary results showed that the best results for Q-learning were obtained with ϵ=0.05. Therefore we will make use of the ϵ-greedy exploration strategy, which means that 95% of the time it selects the action with the highest estimated value, and 5% of the time a random action for exploration purposes.
The difference between the Backward Q-learning algorithm and Monte Carlo learning relies in the learning update. Instead of updating each state the agent visited with the same reward value, the Q-learning algorithm has a more specific update method and takes potential future state values into account. The 'backward' here signifies that the update starts at the last visited state, and from thereon updates each previous state until the first. The backward Q-learning algorithm directly tunes the Q-values, and then the Q-values will indirectly affect the action selection policy. Therefore, this reinforcement learning algorithm can enhance learning speed and improve final performance over the normal Q-learning algorithm (Wang et al. (2013)).
The last action brings the agent in a goal state, and therefore it has no future states. Hence the last state-action pair is updated as follows:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha * (R - Q_t(s,a))$$

This is the same update as the Monte Carlo update. For every other state-action pair but the last one the learning update is given by:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha * (\gamma \max_a Q_t(s',a) - Q_t(s,a))$$

Normally the reward is also a part of the second formula, but it is omitted here for the reason that only the last state-action pair gets an immediate reward.
State-action s is a state and a is a possible action in this state. A state is represented by a state of the network with respect to the values belonging to the role of the agent.
Future-state s' is a potential future state, resulting from an action. Future states for the defender are pretty straightforward since his observable future states are always empty, although allowed actions are dependent on the environmental state, in contrast with the attacker for what future states are limited by the node the agent is currently in.

The best Q-value is taken for the best possible action from future state s'. Thus the highest Q-value of a possible action in a possible next state.
Discount-factor γ is the discount parameter. $\gamma \equiv [0,1]$ and serves to find a balance between future and current rewards. if $\gamma \to 0$ the agent does not consider future rewards at all, while the agent gets more presbyopic if $\gamma \to 1$. In general a high discount factor is used.

3.3 Neural Network

The neural network is implemented only for the defender agent and not for the attacker. Since the attacker only views part of the network at a time, a neural network approach will be highly susceptible to the perceptual aliasing effect (Whitehead and Ballard (1990)).
The neural network uses stochastic gradient descent back-propagation (Russell and Norvig (2003)) to train its weights. It uses an input neuron for each defense or detection value in the network. Since the network contains 4 nodes and there are 10 defense values and 1 detection value per node, this comes down to 44 nodes. Preliminary testing has shown that 6 neurons in the hidden layer connecting to 44 output neurons provides the best results. The 44 output neurons represent each of the 44 moves available to the defender at any game step. The highest activation value amongst the output nodes of the network represents the defensive action that is taken.
Rewards for the neural network are calculated after each game as the Monte Carlo algorithm with rewards normalized to 1 for winning and -1 for losing, using only the selected output per game step as the basis for the stochastic gradient descent algorithm. The actions that lead to winning games are trained to approach a target activation value of 1, while actions contributing to losing are trained to approach -1.

Table 4.1: Parameter settings for the attacker agent that result in the highest win score against an opponent with random action selection

Learning technique	Learning Rate	Parameter(s)
Discounted UCB	$\alpha = 0.05$	$\xi = 4, \gamma = 0.2$
ϵ-greedy	$\alpha = 0.05$	$\epsilon = 0.05$
Q-learning	$\alpha = 0.05$	$\gamma = 0.99, \epsilon = 0.05$
Softmax	$\alpha = 0.3$	$\tau = 1$
UCB-1	$\alpha = 0.05$	$c = 1$

Table 4.2: Parameter settings for the defender agent that result in the highest win score against an opponent with random action selection

Learning technique	Learning Rate	Parameter(s)
Discounted UCB	$\alpha = 0.05$	$\xi = 5, \gamma = 0.2$
ϵ-greedy	$\alpha = 0.1$	$\epsilon = 0.05$
Linear Network	$\alpha = 0.1$	-
Neural Network	$\alpha = 0.1$	# hidden neurons = 6
Q-learning	$\alpha = 0.1$	$\gamma = 0.99, \epsilon = 0.05$
Softmax	$\alpha = 0.4$	$\tau = 1$
UCB-1	$\alpha = 0.4$	$c = 2$

3.4 Linear Network

The linear network follows the same approach as well, but has no hidden layer. It therefore directly calculates its output based on an input layer of 44 nodes, again representing each of the defense and detection values in the network, with weighted connections to the output layer. Each of the outputs represents a possible move in the game. The output with the highest activation is selected as the move to be played. Again the target value of moves contributing to victory are set at 1, while losing moves are set at -1.

4 Experimental Setup

We assumed that the advantage gain for the attacker is equal to the risk he takes, and that the defender wins as much by detecting an attacker as he loses by getting its data stolen. Therefore, the rewards are symmetric. Agents obtain a reward equal to 100 if the agents wins a game, and a reward equal to -100 if the agent loses a game.
To test the performance of the different reinforcement learning techniques, simulations are run in which the attacker and defender agent play the game with different techniques. The attacker plays with six different techniques: random action selection, four different Monte Carlo learning techniques and Q-learning, while the defender plays with eight different techniques, the six from the attacker and the two neural networks.
Every technique is optimized for both agents, by changing the learning rate and its own parameter(s). For optimizing these parameters, simulations are run against an opponent with random action selection, and it is determined which parameter setting results in the best behavior. The optimal parameter settings can be seen in Tables 4.1 and 4.2.
Every technique for the attacker agent is played against every technique for the defender agent, resulting in 48 different combinations. Each combination is simulated 10 times on the standard network described in Section 2, for 20000 games per simulation.

5 Experiments and Results

The results can be seen in Table 5.1, which displays the average win score over the full 10 simulations of 20000 games, and in Table 5.2, which displays the average win score over the last 2000 games of the simulations that ran 20000 games. Both average scores come along with their standard deviations. Each win score is a value between -1 and 1, -1 indicates the attacker has won everything, 1 means the defender did always win. In the bottom row and rightmost column, the best attacker(A)/defender(D) for that column/row is displayed.
From the "average" row and column in both tables, it can be concluded that for both agents there is not a single algorithm that always performs best. Q-learning is a good learning algorithm for the defender, while for the attacker the Monte Carlo learning algorithms with exploration based on confidence bounds show good results.
Plots of the changes in win score during a simulation can be seen in Figures 5.1, 5.3, 5.2, 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9. Each plot shows the average win score over the previous 500 games, with the win score having the same meaning as in Tables 5.1 and 5.2. Figure 5.1 shows that with random action selection the defender wins about 95% of the games, so it has a huge advantage with respect to the attacker. Figure 5.2 shows that, due to this huge

Table 5.1: Average score of complete simulations of 20000 games for each strategy pair along with the standard deviation. DUC = discounted UCB, GRE = ϵ-greedy, LN = linear network, NN = neural network, QL = Q-learning, RND = Random, SFT = Softmax, UCB = UCB-1. The average row/column shows the total average score for each attacker/defender strategy against all opponents.

D ↓ A →	DUC	GRE	QL	RND	SFT	UCB	average	BEST A
DUC	-0.25 ± 0.11	-0.28 ± 0.10	-0.05 ± 0.22	0.93 ± 0.02	-0.24 ± 0.13	-0.34 ± 0.12	-0.04 ± 0.48	UCB
GRE	-0.17 ± 0.04	-0.16 ± 0.03	0.13 ± 0.10	0.97 ± 0.005	-0.08 ± 0.05	-0.14 ± 0.07	0.09 ± 0.44	DUC/GRE
LN	-0.96 ± 0.02	-0.91 ± 0.02	0.00 ± 0.90	0.94 ± 0.02	-0.90 ± 0.11	-0.89 ± 0.19	-0.45 ± 0.77	DUC
NN	-0.61 ± 0.20	-0.56 ± 0.24	-0.32 ± 0.53	0.93 ± 0.03	-0.07 ± 0.59	-0.60 ± 0.26	-0.21 ± 0.59	DUC/GRE/UCB
QL	-0.14 ± 0.03	-0.17 ± 0.01	0.28 ± 0.06	0.94 ± 0.003	-0.05 ± 0.04	-0.15 ± 0.03	0.12 ± 0.44	GRE
RND	-0.96 ± 0.01	-0.92 ± 0.01	-0.92 ± 0.005	0.92 ± 0.002	-0.96 ± 0.01	-0.96 ± 0.01	-0.63 ± 0.76	DUC/SFT/UCB
SFT	-0.43 ± 0.08	-0.36 ± 0.05	-0.01 ± 0.11	0.97 ± 0.01	-0.27 ± 0.11	-0.43 ± 0.09	-0.09 ± 0.54	DUC/UCB
UCB	-0.64 ± 0.15	-0.67 ± 0.08	-0.47 ± 0.16	0.95 ± 0.01	-0.54 ± 0.20	-0.54 ± 0.18	-0.32 ± 0.63	DUC/GRE
average	-0.52 ± 0.33	-0.50 ± 0.31	-0.17 ± 0.38	0.94 ± 0.02	-0.39 ± 0.37	-0.51 ± 0.31		DUC/GRE/UCB
BEST D	QL	GRE/QL	QL	SFT/GRE	NN/QL	GRE/QL	GRE/QL	

Table 5.2: Average score of last 2000 games for each strategy pair along with the standard deviation. DUC = discounted UCB, GRE = ϵ-greedy, LN = linear network, NN = neural network, QL = Q-learning, RND = Random, SFT = Softmax, UCB = UCB-1. The average row/column shows the total average score for each attacker/defender strategy against all opponents.

D ↓ A →	DUC	GRE	QL	RND	SFT	UCB	average	BEST A
DUC	-0.35 ± 0.17	-0.33 ± 0.15	-0.18 ± 0.19	0.94 ± 0.03	-0.31 ± 0.25	-0.54 ± 0.14	-0.13 ± 0.54	UCB
GRE	-0.20 ± 0.06	-0.16 ± 0.08	-0.10 ± 0.09	0.98 ± 0.01	-0.14 ± 0.19	-0.17 ± 0.13	0.04 ± 0.46	DUC
LN	-1.00 ± 0.00	-0.96 ± 0.01	0.00 ± 1.03	0.93 ± 0.03	-1.00 ± 0.00	-0.93 ± 0.22	-0.49 ± 0.80	DUC/SFT
NN	-0.56 ± 0.33	-0.56 ± 0.24	-0.12 ± 0.62	0.91 ± 0.02	-0.06 ± 0.74	-0.70 ± 0.32	-0.18 ± 0.59	UCB
QL	-0.10 ± 0.10	-0.19 ± 0.05	0.28 ± 0.12	0.94 ± 0.01	-0.03 ± 0.18	-0.12 ± 0.001	0.13 ± 0.43	GRE
RND	-0.97 ± 0.01	-0.94 ± 0.005	-0.93 ± 0.01	0.93 ± 0.004	-0.93 ± 0.08	-0.97 ± 0.005	-0.64 ± 0.77	DUC/UCB
SFT	-0.55 ± 0.23	-0.41 ± 0.15	-0.08 ± 0.15	0.98 ± 0.01	-0.46 ± 0.35	-0.51 ± 0.20	-0.17 ± 0.59	DUC/UCB
UCB	-0.73 ± 0.29	-0.77 ± 0.22	-0.74 ± 0.46	0.95 ± 0.02	-0.45 ± 0.25	-0.56 ± 0.26	-0.38 ± 0.66	DUC/GRE/QL
average	-0.56 ± 0.33	-0.54 ± 0.32	-0.23 ± 0.40	0.95 ± 0.02	-0.42 ± 0.37	-0.56 ± 0.31		DUC/GRE/UCB
BEST D	QL	GRE	QL	GRE/SFT	NN/QL	QL	GRE/QL	

advantage, the ϵ-greedy algorithm for the defender has not much room for improvement and it hardly improves at all. Figure 5.3 shows that the ϵ-greedy attacker learns very fast to attack the gaps in the network. He learns so quickly because of the gaps in the network, which results in some actions being much better than others.

Figure 5.4 shows the behavior of the agents when both learn using ϵ-greedy. The win rate shows a lot of fluctuations, which means the agents try to optimize their behavior by adapting to each other, and overall the attacker optimizes slightly better than the defender, resulting in an average win rate of around -0.1.
However, not all algorithms can adapt this quickly to each other. Figure 5.5 shows that the defender, which learns using UCB-1 exploration, has troubles reacting to changes in the attacker's behavior. It takes a long time before the defender comes up with a good answer. However the attacker can quickly adapt to this new behavior, which again leaves the defender with a for him hard problem to solve. This behavior is due to the UCB-1 exploration algorithm being unsuitable for non-stationary environments like the simulation in this paper. However, sometimes it comes up with a good answer, an example can be be seen in Figure 5.6 in which the defender after a while behaves in such a way that the win score stays around -0.3 instead of almost -1.0.
In Figures 5.7 and 5.8 it can be seen that the Discounted UCB exploration algorithm optimizes the defender agent much better than the UCB-1 algorithm. There are no parts in which the defender wins almost no game at all, so the adaptations go much better here, although a fall like in Figure 5.7 is still possible. This difference can also be seen in Tables 5.1 and 5.2.
The defender agent playing with Discounted UCB has against every opponent except for the random attacker a better win score than the defender

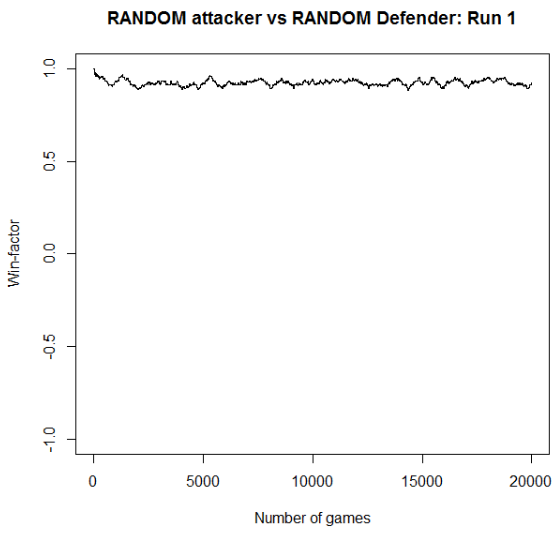

Figure 5.1: Average score of previous 500 games for Random against Random

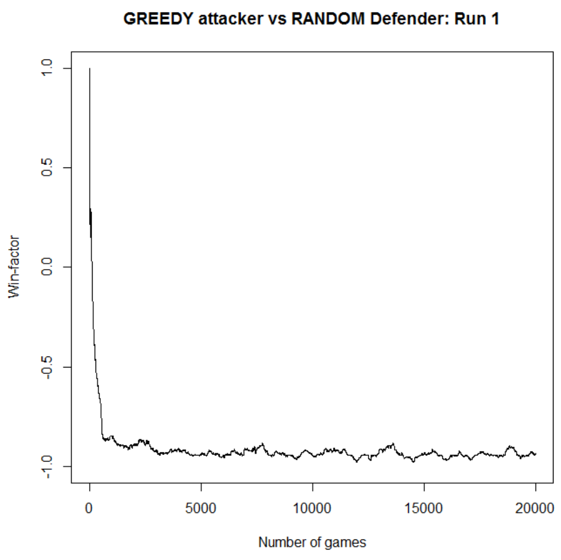

Figure 5.3: Average score of previous 500 games for ϵ-Greedy against Random

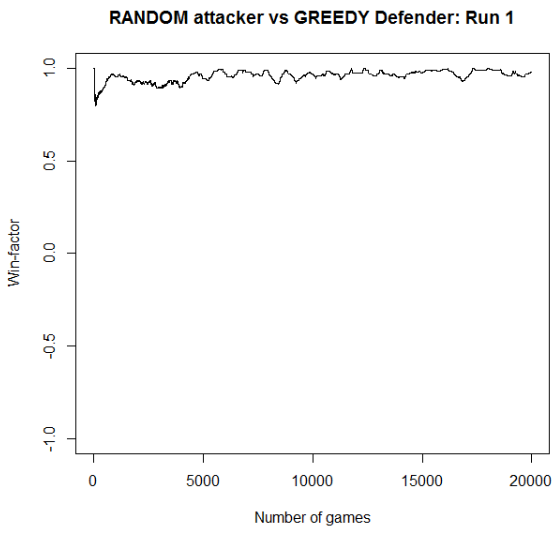

Figure 5.2: Average score of previous 500 games for Random against ϵ-Greedy

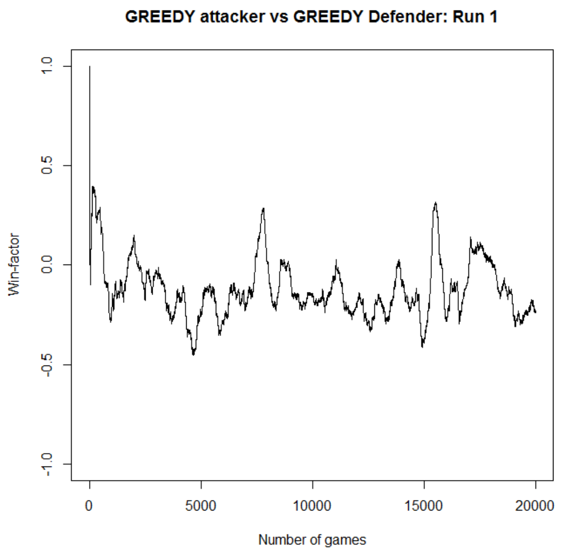

Figure 5.4: Average score of previous 500 games for ϵ-Greedy against ϵ-Greedy

agent with UCB-1. This supports the argument made in Garivier and Moulines (2008) that the Discounted UCB algorithm is better than UCB-1 in non-stationary environments like the simulation in this paper. The (slightly) better score for UCB-1 against a random opponent strengthens this prove, because due to the randomness of the attacker the defender automatically wins a lot (see the score for random against random) and so the environment is a lot more stationary.

For the attacker playing with Discounted UCB

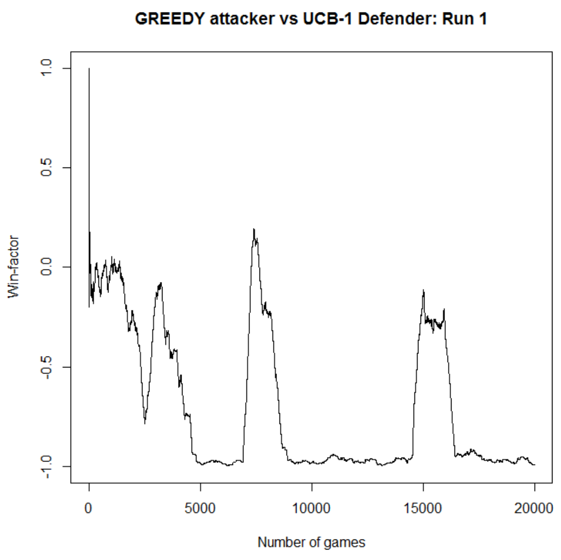

Figure 5.5: Average score of previous 500 games for ϵ-Greedy against UCB-1

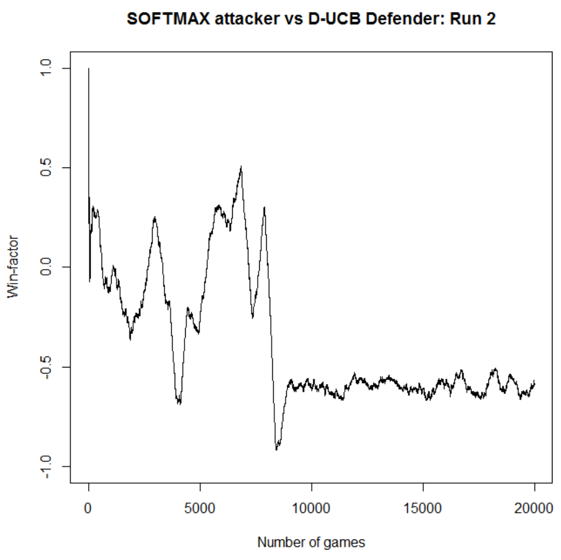

Figure 5.7: Average score of previous 500 games for Softmax against Discounted UCB

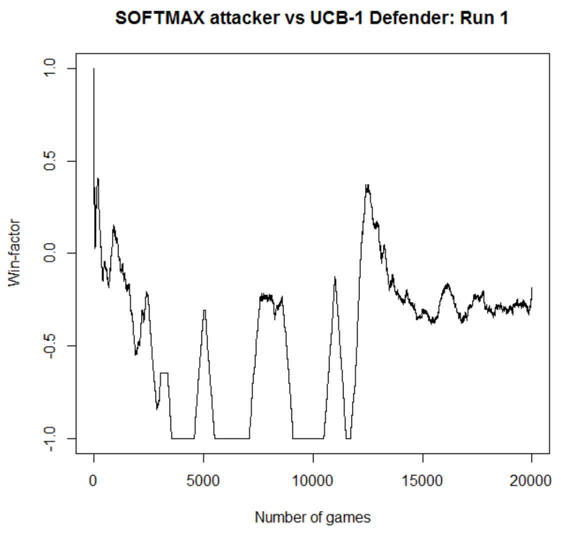

Figure 5.6: Average score of previous 500 games for Softmax against UCB-1

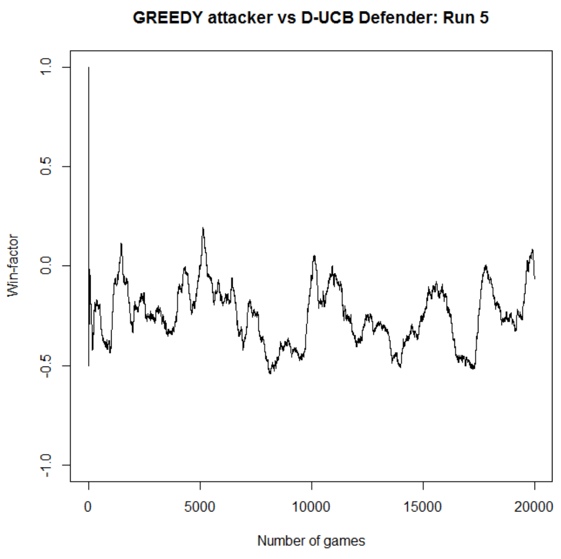

Figure 5.8: Average score of previous 500 games for ϵ-Greedy against Discounted UCB

there is almost no difference in results with respect to the attacker playing with UCB-1. Both are among the best performing algorithms. This could indicate that for the attacker the environment is less non-stationary than for the defender.

The Q-learning defender has relatively good results compared to the Softmax and UCB algorithms. The final scores from Table 5.1 fluctuate around those from the ϵ-greedy defender. Remarkable is the Q-learning defender versus the Q-learning attacker

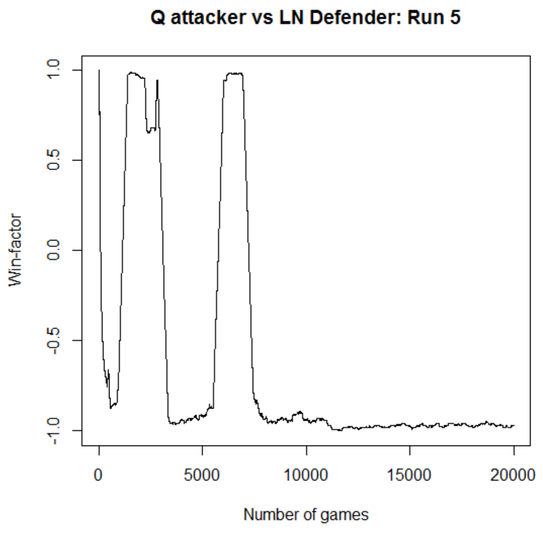

Figure 5.9: Average score of previous 500 games for Q-learning against Linear network

where the defender does extremely well compared to the other defenders.
We also observed that the Q-learning defender does well in the last 2000 games. A possible explanation for this phenomenon could be that Q-learning learns well on the long run.
The Q-learning attacker does worse than other algorithms. This is probably the case because the action-chains, or optimal policy is less clear for the attacker. The next state is also dependent whether his action succeeded or not, whereas the defender has a fixed network with increasing defense values. Another possible explanation for the Q-learning attacker under-performing the Monte Carlo ϵ-greedy algorithm could be that Q-learning takes slightly longer to find an optimal policy, while the standard update learns action sequences with a reasonable outcome faster. This also supports the earlier study from Szepesvári (1997), who found that Q-learning may suffer from a slow rate of convergence, especially when the discount factor is close to one. The last-mentioned is crucial in an environment with a changing optimal policy. The difference in convergence between both agents could be the result of the properties of the network, leading to more logical rational actions for the defender.
Unique are the results from the Q-learning attacker against the linear network. Figure 5.9 shows one of the simulations. We observed really high fluctuations, which also explains a standard deviation of 0.90.
The neural network performs worse than most of the other defender algorithms. It also has high fluctuations in its performances, sometimes managing a 50% and at other times dropping as low as 10%. This discrepancy is due in part to the problem of local optima for gradient descent algorithms (Schmidhuber (1994)). By becoming stuck in a local minimum the neural network is unable to move to a more advantageous strategy. This explains the wild fluctuation in results as the starting position of its weights is random.
The results also reflect the problem that neural networks have with adversarial learning. The network is slower in adapting to a constantly changing environment than other algorithms, therefore being consistently beaten by attackers. Lastly the stochastic nature of the detection parameter means that the succes of the network depends on chance as well. This obstacle may affect the neural network more strongly than a tabular approach. The neural network performs backpropigation for each step in the game, therefore repeatingly appyling the result of one game to the whole network. It might therefore be much more affected by an early game fluke than a tabular based approach. The linear algorithm consistently fails against all other algorithms except for the Q-learning algorithm (see Figure 5.9). This could be explained by the tendency of both algorithms to get stuck in local optima, taking a long time to recuperate and find a different solution. This is also reflected in the results. We can see one of the two algorithms being consistently dominant until a switching point, after which the other algorithm completely dominates.

5.1 Detection Actions

The first thing the defender agent has to do is fixing at least the security flaw in the end node. After that, the agent must choose between an investment in defense or in detection. The former will in essence only postpone a successful attack, while the latter increases the actual win chance of the defender when an attack is blocked. It is therefore investigated if more investments in detection leads to a higher win rate for the defender. For the runs

Table 5.3: Average number of detection actions of defender against ϵ-greedy attacker per game, along with the standard deviation

Defender algorithm	Total average	Last 2000 average
Discounted UCB	0.30 ± 0.09	0.44 ± 0.36
ϵ-greedy	0.33 ± 0.04	0.31 ± 0.16
Linear Network	0.08 ± 0.01	0.00 ± 0.00
Neural Network	0.33 ± 0.19	0.35 ± 0.20
Q-learning	0.19 ± 0.06	0.20 ± 0.21
Random	0.20 ± 0.004	0.20 ± 0.01
Softmax	0.31 ± 0.03	0.34 ± 0.13
UCB-1	0.24 ± 0.07	0.20 ± 0.06

against an attacker using Monte Carlo learning with ϵ-greedy exploration, it is counted how many detection investment actions the defender performs per game. The results are displayed in Table 5.3. None of the algorithms has a significantly higher average number of detection actions than all others, and two of the four algorithms with a (slightly) higher average are not among the best defenders. It can be concluded from this that more detection investments does not automatically lead to a higher win rate. The averages over the last 2000 games are in general a bit higher, which might indicate that detection investments are a good thing to do, but only if the agent knows that the defense will probably hold.

6 Conclusion

In this paper we made a cyber security simulation with two learning agents playing against each other on a cyber network. The simulation is modelled as an extensive form game with incomplete information in which the attacker tries to hack the network and the defender tries to protect it and stop the attacker. Monte Carlo learning with several exploration strategies (ϵ-greedy, Softmax, UCB-1 and Discounted UCB) and Q-learning for the attacker and two additional neural networks using stochastic gradient descent back-propagation for the defender (one linear network and one with a hidden layer) are evaluated. Both agents needed to use the algorithms to learn a strategy to win as many games as possible, and due to the competition the environment became highly non-stationary. The results showed that the neural networks were not able to handle with this, but the Monte Carlo and Q-learning algorithms were able to adapt to the changes in behavior of the opponent. For the defender, Q-learning and Monte Carlo with ϵ-greedy exploration performed best, while for the attacker Monte Carlo learning with discounted-ucb exploration, with ucb-1 exploration, and with ϵ-greedy exploration obtained the best results.

7 Discussion

Taking the cyber security game described in this paper as a base, there are several parts that could be researched further.
An interesting experiment could be run with an extended network made up out of more nodes. In addition, such an extended network could also contain more than one asset node, requiring the attacker to find one or all of them. In theory these algorithms could handle these different situations as well.
The network could also be expanded by creating even more attack types, making the defense more complex. We expect this to favor the attacker in winning the game and making the defender more likely to focus on the detection parameter. The attacker would probably profit from the increased number of avenues of attack making it harder for the defender to predict where the attacker will attack. Therefore the defender will most likely prefer the detection parameter, which will work against any attack.
More agents could also be added to the network. Different attacker agents attacking the network simultaneously can train the defender to divide resources, simulating the realities of a large online network or event. A large network could also be governed by multiple neural network defenders, each responsible for a small sector, governed by a macro network or other optimization strategy.
Better results could be achieved with the Q-learning algorithm by combining it with various exploration techniques, as has been done with the Monte Carlo algorithms.
The neural network could be expanded by using different exploration algorithms to improve its ability to avoid a local minimum. It could also benefit from slowing down its reaction to change, perhaps by not using every game step to update or by using

a combination of past results in the update function. More research in partially observable adversarial learning problems could definitely improve the functionality of neural networks. Future work also includes adding different reward models. The current reward is an all or nothing approach. Different rewards could be added by for example adding a small reward if the attacker intrudes another node.

8 Contributions of Members

The basic game is implemented by the three of us. Richard and Leon concentrated on the game itself and the layout, while Albert concentrated on the implementation of the agents with random action selection. The Monte Carlo learning algorithm with ϵ-greedy exploration was initially taken as the basic learning algorithm to compare others with, so this is also implemented by the three of us.
Then Richard concentrated on the implementation of the various exploration algorithms for Monte Carlo learning, Leon focused on the Q-learning algorithm, and Albert implemented the two neural networks. The experiments are divided as follows: Richard performed all experiments involving only Monte Carlo learning algorithms, Leon did all experiments involving Q-Learning, and Albert did, with a little help of the other two, the experiments involving the neural networks.
The text of this thesis is divided as follows: the abstract is made by Albert and Richard. The Introduction is made by the three of us: Albert focused on general introduction and previous work, while Richard and Leon wrote the contributions part and Richard wrote the outline. The Cyber Security game section is for the biggest part written by Richard, except for the standard network part and the two figures, which are made by Leon. In the Reinforcement Learning section, the general part, Monte Carlo Learning, and Exploration Algorithms are written by Richard, Q-Learning is written by Leon and Neural Network and Linear Network are written by Albert. The Experiments and Results section is, up to the discussion about the performance of Q-learning, written by Richard, the Q-learning discussion itself is written by Leon, the discussion about the neural networks is written by Albert, and the part about detection actions is written by Richard. Richard also wrote the conclusion, while Albert and Leon wrote the part about future work. The tables are "designed" by Richard and filled in by the three of us, and the plots were also made by the three of us.

References

The open web application security project. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. Accessed: 2016-08-11.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

D.A. Berry and B. Fristedt. *Bandit Problems: sequential allocation of experiments.* Chapman and Hall, London/New York, 1985.

K. Chung, C. Kamhoua, K. Kwiat, Z. Kalbarczyk, and K. Iyer. Game theory with learning for cyber security monitoring. *IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pages 1–8, 2016.

A. Garivier and E. Moulines. On upper-confidence bound policies for non-stationary bandit problems. *Proc. of Algorithmic Learning Theory (ALT)*, 2008.

L. Huang, A. Joseph, B. Nelson, B. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of 4th ACM Workshop on Artificial Intelligence and Security*, pages 43–58, 2011.

M. Lopes, T. Lang, M. Toussaint, and P.Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 206–214. Curran Associates, Inc., 2012.

A. Mahajan and D. Teneketzis. *Multi-Armed Bandit Problems*, pages 121–151. Springer US, Boston, MA, 2008.

R. Myerson. *Game Theory.* Harvard University Press, 1991.

J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior.* Princeton University Press, 2007.

S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu. A survey of game theory as applied to network security. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10, 2010.

S. Russell and P. Norvig. *Artificial intelligence a modern approach*. Prentice Hall, New Jersey, 2003.

J. H. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. San Mateo, CA: Morgan Kaufmann, 1991.

J.H. Schmidhuber. Discovering problem solutions with low Kolmogorov complexity and high generalization capability. Technical Report FKI-194-94, Fakultät für Informatik, Technische Universität München, 1994.

A. Sharma, Z. Kalbarczyk, J. Barlow, and R. Iyer. Analysis of security data from a large computing organization. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 506–517. IEEE, 2011.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book, 1998.

C. Szepesvári. The asymptotic convergence-rate of q-learning. In *NIPS*, pages 1064–1070. Citeseer, 1997.

S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University, January 1992.

Y.H. Wang, T.H.S. Li, and C.J. Lin. Backward q-learning: The combination of sarsa algorithm and q-learning. *Eng. Appl. of AI*, 26:2184–2193, 2013.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

S.D. Whitehead and D.H. Ballard. Active perception and reinforcement learning. *Neural Computation*, 2(4):409–419, 1990.

M.A. Wiering and J.H. Schmidhuber. Efficient model-based exploration. In J. A. Meyer and S. W. Wilson, editors, *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 6*, pages 223–228. MIT Press/Bradford Books, 1998.

M.A. Wiering and M. van Otterlo. *Reinforcement Learning: State of the Art*. Springer Verlag, Berlin Heidelberg, 2012.