Complexity and problem solving-
A tale of two systems

Marcus Andersson
I want to thank everyone related for making this thesis possible, my supervisor Kai-Florian Richter who has been very helpful. A special thanks to my family and friends.
Abstract

The purpose of this thesis is to investigate if increasing complexity for a problem, makes a difference for a learning system with dual parts. The dual parts of the learning system are modelled after the Actor and Critic parts, from the Actor-Critic algorithm, using the reinforcement learning framework. The results conclude that not any difference can be found in the relative performance in the Actor and Critic parts, when increasing the complexity of a problem. These results could depend on technical difficulties in comparing the environments and the algorithms. The difference in complexity would then be non-uniform in an unknowable way and uncertain to use as comparison. If on the other hand the change of complexity is uniform, this could point to the fact that there is an actual difference in how each of the actor and critic handles different types of complexity. Further studies with a controlled increase in complexity are needed to establish which of the scenarios is most likely to be true. In the discussion an idea is presented of using the Actor-Critic framework as a model to understand the success rate of psychological treatments better.

Keywords: Complexity, Problem solving, Actor-Critic, Reinforcement learning

Abstrakt


Nyckelord: Komplexitet, Problemlösning, Actor-Critic, Reinforcement learning
Complexity and problem solving -
A tale of two systems

Even though the art of problem-solving has been relevant since the inception of mankind, recent developments has put some aspects of it into a new light. Developments in artificial intelligence have led to computer programs which can solve complex tasks previously thought only humans could process.

For example, AlphaGo, a computer program made by DeepMind in 2016 beat several world champions in the board game of go, that was previously thought impossible for a computer program to do (Silver et al., 2017).

In light of these developments, the aim and purpose of this project is to investigate if increasing the complexity between problems, makes a difference for a learning system with dual interacting parts. These dual acting parts works in tandem together to improve and solve a problem. In this thesis these dual parts are called Actor and Critic (see more in the sections ”Dual systems in human cognition” and “Actor-Critic algorithms” later in this introduction). The experiments are made with an artificial simulation of a learning system with dual acting parts using the reinforcement learning framework.

### Complexity and problem-solving

What is considered a complex problem is up for debate (Dörner & Funke, 2017). A complex problem is generally seen as a problem with several moving interdependent parts of which interactions affect the system in an often unforeseeable way (Kurtz & Snowden, 2003).

An interesting way to look at complex problems is by the cynefin framework (Kurtz & Snowden, 2003). The cynefin framework created by David Snowden, divides the solving of problems into four different categories: simple, complicated, complex and chaotic problems. The solutions to simple problems are often obvious with known best practices, while complicated problems requires more knowledge, as well as the use of expert knowledge and uses good practices.

Complex problems do not generally have a clear immediate solution, the cause and effect of the system with a complex problem, is often unclear and requires experimentation to solve. When experimenting with a complex system, it is possible to find patterns that emerge, and the pattern can then be used to find a strategy or code for solving the complex problem. When a solution is known or the underlying problem changes, it is possible to move between the categories of the cynefin framework (Kurtz & Snowden, 2003).

In early complex problem-solving research in the 1970’s, a problem simulated with 20 interdependent variables was considered to be complex, while modern research can use simulations with more than 2000 variables. A consensus for the minimum number of interdependent variables for a complex system seems to not have been established. While the consensus is that the upper bound of the number of variables for complex problems is unbound (Dörner & Funke, 2017).
Reinforcement learning theory

Reinforcement learning (RL) is a branch of machine learning that is concerned with how an agent learns and solves a problem in a given environment. One of the fundamental theories behind reinforcement learning is Markov decision processes. Reinforcement learning functions so that at each step the agent of the environment is in some state $s$. From that state the agent can choose any action $a$ that is possible from that state. Depending on the action the agent makes, combined with stochastic forces in the environment, the agent ends up in a new state $s'$. In that new state $s'$ the agent also gets a reward $R_a(s, s')$ for being in that state. Using that reward it is possible to calculate how valuable the action leading to that state is (Sutton & Barto, 2017).

Different families of RL

It is possible to divide reinforcement learning algorithms into two different groups, depending on if the algorithms optimize the policy function or the value function, of the solution for the problem. Algorithms that optimize the policy function directly try to parameterize and figure out the best actions possible. While algorithms that optimize the value function instead finds the optimal value for each state, so that the policy indirectly can choose the policy with the highest value.

For example, if you play a car racing game, a policy function would directly optimize how the controls would have to shift on the game controller, to get the best score in a game. While a value function would first find out how good or bad all the game states are. After the state values are known, the value function selects the move that would take the agent to the state with the highest value. There is also a third type of reinforcement learning algorithm that combines both policy optimization and value functions algorithms into one algorithm, called Actor-Critic methods (Sutton & Barto, 2017).
Figure 1. From https://planspace.org/20170830-berkeley_deep_rl_bootcamp/. The figure shows different types of reinforcement learning algorithms: policy optimization and value function algorithms. Dynamic programming is here the same as value function methods. In this thesis, policy gradient is the only method used from the policy optimization methods.

Value function algorithms

Value function type algorithms try to estimate the value of how much each state in an environment is worth. If you for example play a car racing game, a value function algorithm is trying to estimate how much every single frame in the video game is worth, and since you have at least 24 frames a second for a video game, the state space is usually an enormous space to estimate from if you have hours of gameplay to sample from. This is made simpler by value function approximators such as neural networks that generalizes and compresses the state space for a more manageable computation. It is advances in value function approximators, and the increase in computational power that have made the recent successes in reinforcement learning possible. When the value estimations are done, it is possible to select actions greedily, leading to the highest value states for the best actions possible (Sutton & Barto, 2017).

\[
V_\pi(s) = E[R] = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s],
\]

Equation 1. Value formula for a state

The above formula (Equation 1), is the formula for calculating the value of one given state. The formula calculates the value of a certain state, that state is above described as the zeroth state \(s_0=s\). The value of the state \(V_\pi(s)\) is equal to the expected reward \(R\), which is the sum of all the following actions and rewards taken after that action. That sum is also discounted by a factor \(\gamma_t\). This discounting factor \(\gamma_t\) means that states closer to the original state are given a higher weight compared to states farther away from it. The reward \(R\) is a signal that indicates how advantageous the state is for solving the current problem in a given environment. The reward \(R\) can be calculated from multiple sources such as overall success rates or game scores. \(\pi\) signifies that a certain policy is used. Usually the best way for a policy to act, is to act greedily and select the action that have the highest value \(V\) at each timestep (Sutton & Barto, 2017).

\[
V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))
\]

Equation 2. Value update formula for a state

The formula above (Equation 2) is a learning process update for the value function in reinforcement learning. This means that if an agent goes from state \(s\) to \(s'\), it will update the old state value \(V(S_t)\), with the difference between what the agent beforehand thought the value would be \(V(S_t)\), and the return of actual real value of the new state \(G_t\). This difference is multiplied by the learning rate \(\alpha\) to make each update of the learning process smoother.

For example if the agent on beforehand thought that going from a state \(s\) to another state \(s'\) would be worth 10 in reward, and when the agents gets to \(s'\) it discovers that the state
is actually worth 8, this update would in this case look like this:

\[ \text{Update value} = 10 + (8-10) \text{(Sutton & Barto, 2017)} \]

A classic environment for illustrating reinforcement learning algorithms is the mountain car environment. The goal of the mountain car environment is to get up the hill with a car, but from the start you cannot get up the hill right away, because the power of the car’s engines is insufficient. For succeeding the agent has to learn how to use gravity and to go backwards to create a swing-like effect to get up the hill.

\[ \nabla_{\theta} J(\theta) = \mathbb{E}_{\pi(\theta)}[\nabla_{\theta} \log \pi(\theta) \, Q(\theta(s,a))] \]

Equation 3. Policy Gradient theorem

Figure 2. From http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html. A 3d representation of the value function space of an optimal policy to get up the hill in the mountain car environment. The X-dimension describes the position where the mountain car is, the Y-dimension describes how much velocity should be used, and the Z-dimension describes how much reward a position-velocity pair is worth. Note that the best policy is clear to see, and has a distinct shape.

Policy gradients

For policy gradient type algorithms, the goal is to directly parameterize and improve the policy directly, instead of parameterizing it indirectly as value function methods do. In this thesis policy gradients are the only type of method used from the policy optimization methods. An advantage of policy gradient methods is that it can handle stochastic policies, which pure value function methods cannot do.
improved policy. This is done by sampling an expectation $E_{\pi \theta} \log \pi_{\theta}(a|s)$, an objective function acts like a tracker of how good or bad the policy is. The reason why the policy $\pi$ is in logarithmic form, is due to a statistical method called maximum likelihood estimation (MAE). This is multiplied by the estimated value of the state $Q_{\pi \theta}(s,a)$ to get the actual estimated reward. (Sutton and Barto, 2017)

To do an update of the policy function and start learning of the reinforcement agent, the most basic policy gradient algorithm that can be used is called Monte-Carlo Policy Gradient (REINFORCE). The REINFORCE algorithm is basically the policy gradient theorem (Equation 3) seen above, multiplied by a learning rate to smoothen out the learning and avoid overshoot of local maxima.

$$\Delta \theta_t = \alpha \nabla \theta \log \pi_{\theta}(s_t,a_t)v$$

Equation 4. Update of the policy gradient theorem REINFORCE

The REINFORCE formula seen above (Equation 4), which is same as the policy gradient theorem, with the same notations used (Equation 3), with one exception in the notation for $v$, which is the return of state and is used for the action value $Q$ as an estimate for the state value. The return value $v$ is in this case the combined reward at the end of one episode. The difference between the general policy gradient theorem and REINFORCE, is then that the policy gradient theorem estimates the discounted state value for all the expected future episodes, while REINFORCE in its basic form updates and uses the values of states from the current episode only (Sutton & Barto, 2017).

Actor Critic algorithms

The Actor-Critic algorithms are similar to the policy gradient algorithm REINFORCE, but instead of the one episode estimate $Q_{\pi \theta}(s,a)$ as seen in the policy gradient theorem (Equation 3), or the $v$ value as used in REINFORCE (Equation 4). Actor-Critics instead use an estimator function $Q_{\pi \theta}(s,a)$ for the state value, which is approximating the real value for that state, $Q_{w}(s,a) \approx Q_{\pi \theta}(s,a)$. For the gradient theorem formula the estimator function $Q_{\pi \theta}(s,a)$ is called the critic part, while $\log \pi_{\theta}(a|s)$ is the policy gradient part and is called actor, thus the name Actor-Critic.

The critic part $Q_{\pi \theta}(s,a)$ is similar to the value function methods, in fact it is in practice the same as value function methods. This means that Actor-Critic algorithms are a combination of both policy gradients and value function methods.

The idea is that the policy gradient improves the policy and the critic part in turn gives an estimate of how good the policy is. Both the actor and the critic are parameterized and updated with feedback, to improve the policies as well as the estimates of the policy (Sutton & Barto 2017).

Difference between algorithms in practical computation

The different types of algorithms, policy gradient and value function have been around for some time (Sutton & Barto 2017). During that time the performance has been tested against
each other using the same environment. For example, in one experiment that compared the two types, with implementations that were state of the art in the year 2006, found that the policy gradient algorithm learnt to solve the problem much faster. The value function algorithm on the other hand was much slower in increasing the performance at the beginning of training, but continued to learn throughout the experiment, and at the end of the experiment it was performing significantly better than the policy gradient algorithms (Beitelspacher et al 2006). These findings are contraindicated in today’s literature, where it is assumed that value function algorithms generally are more data-efficient. There are however some types of problems that value-function algorithms are not applicable to use for, which instead policy gradients can solve (Sutton & Barto 2017).

![Figure 3. Policy gradient vs. value function algorithms. From Beitelspacher et al (2006). The chart above shows the performance of different algorithms in the Spacewar environment. The X-axis shows how many episodes of training have elapsed, and the Y-axis shows the average lifetime, which is how well the algorithms performs in an increasing scale from 0–200. The drawn lines represent different strategies that the player can take such as DoNothing, Hide, Seek and the two algorithms. The OLGARB algorithm in red is a policy gradient algorithm, and the Sarsa(λ) in blue, is a value function algorithm. It can be seen that initially the policy gradient algorithm increases in performance faster. While the value function algorithm shown in blue continues to increase in performance, and outperforms the policy gradient at the end of training.]

**Dual systems in human cognition**

There are some examples of dual acting systems found in human cognition and decision-making processes. Kahneman (2011) suggests that humans have a fast, automatic and unconscious system called system 1 for making quick estimates and decisions. Further we also have a system 2 that is used for slow, conscious, deliberate reasoning and capable of analyzing decisions in a logical manner.
The dual processing theory (Evans, 2008), is a theory that unifies many separate findings of dual acting systems found in cognition and psychology research, into two similar categories as proposed by Kahneman (2011).

Furthermore, recent findings in higher order attention suggest we have two types of attention, one is focused attention which handles focused tasks and is adept at ignoring distractions, while diffuse attention is better at handling creative problem solving and finding patterns (Amer et al., 2016). Comparing the findings of Amer et al. (2016) to the dual processing theory of Evans (2008), seems not directly applicable, due to that both the attention systems per se is conscious. Even though there still seems to be similarities between the theories, for example, the two systems in both theories have similar categories for problem-solving (Amer et al., 2016; Evans, 2008).

![Figure 4. From (Amer et al., 2016). The figure shows the correlation between the level of cognitive control as seen in the X-Axis. On the Y-axis the degree of performance is seen. Diffuse attention shown in blue generally uses low cognitive control, while focused attention shown in red uses high level of cognitive control. Note that each type of attention has a strong suit of tasks that it is most proficient in.](image)

**Measuring the performance and hypothesis**: The hypothesis in this report is that with increasing complexity of a problem the different parts of an Actor-Critic algorithm will perform differently. This performance is measured with how well an agent completes the problem in front of them, this can for example be the “score” at the end of a given task. By recording the performance values, it is possible to see if any of the Actor or Critic performed better or worse, compared to each other in a given environment.
Method

In order to perform a comparison of the different parts of the Actor-Critic algorithms under different levels of complexity, both the algorithms and the environments need to be chosen. Here, algorithms will be tested in two different environments, and three algorithms will be compared to each other.

Setting up the environment and the algorithms

Choosing the environments: To test the hypothesis if increasing complexity makes any difference for the parts in an Actor-Critic algorithm, two environments were chosen. The two environments are one with low complexity, and one environment with high complexity to be able to test for the difference in complexity.

When choosing an environment with low complexity, the cliff walking environment from Sutton & Barto (2017) was chosen. The cliff walking environment have few possible actions, and few state variables making both the values and actions low in complexity. The cliff walking environment starts the agent at grid location $S$ and the goal is to go tile $G$ where the maximum reward is. For each step the agent loses -1 in reward and if the agent goes to a grid area called the cliff, it gets -100 in reward. Depending on the algorithm used the agent usually starts learning by going a far way around the cliff to avoid the big penalty of -100, but eventually learns the quickest way right above the cliff.

![Figure 5. The cliff walking environment from (Sutton and Barto, 2017). In the cliff walking environment, the performance is measured on how many points the agent has when it gets to the goal.](image)

For the high complexity environment, the Vizdoom environment from the Vizdoom team (2018) was chosen. The Vizdoom environment have a high complexity in both the action space and the state value space. Vizdoom is a simplified version of the game DOOM for testing reinforcement learning. In this scenario the goal is to defend the center. The player can only rotate to look left or right, as well as shoot or not shoot. Monsters spawn randomly and start approaching the player and can harm the player if not dealt with. The amount of ammunition started with is also limited to 25 shots.
Figures 6. From playing Vizdoom (Vizdoom team, 2018) The goal is to survive and defend the center. The player can only rotate to look left or right, and shoot or not shoot. Monsters spawn randomly and start approaching the player and can harm the player if not dealt with. The performance is measured as to how well the player survives.

**Choosing the algorithms:** To be able to have a comparability between the environments, the same types of algorithms should be used in both environments. Depending on how the environments are and which actions that is possible to make, the implementations of the algorithms might look different, but the different algorithms should still use the same core mechanics to be able to be comparable between the environments. As a selection criterion for the algorithms, it was chosen to use as simple as possible algorithm for each category, to make comparisons as simple as possible. For both environments REINFORCE was used, which is the most basic algorithm for the actor-part and often used as a benchmark for policy gradients (Sutton and Barto, 2017). For the Critic-part, a Q-learning algorithm and a DDQN(Double-Deep Q-learning) algorithm was used. DDQN is a slightly more advanced version of Q-learning, and both algorithms are commonly used value-function algorithms (Sutton and Barto, 2017). In both the environments an Advantage Actor-Critic algorithm was used, which is a basic type of Actor-Critic algorithms (Sutton and Barto, 2017). In the Vizdoom environment an Actor-Critic algorithm with a LSTM feature (Long short-term memory) (Hochreiter & Schmidhuber, 1997) was tested, to investigate if increasing the sequential memory buffer would have any difference for the efficiency of the Actor-Critic algorithm.

**Implementations of the algorithms**

Different algorithms and implementations were used depending on the environment. The goal was to get one algorithm from each subgroup of the reinforcement learning algorithm family. In this case that that means using one Policy Gradient, one Actor-Critic and one Value function algorithm.

**The Cliff Walking environment.** The algorithms used for the Cliff walking environment were REINFORCE, Advantage Actor-Critic and DDQN. The algorithms used are the same as found in the reference material for the book Reinforcement Learning: An Introduction by (Sutton & Barto, 2017). The environment Cliff Walking can also be found in the same material. Running the algorithms took about 1–2 minutes for 2000 episodes on a standard computer. The source code used for the agents can be found at “https://github.com/dennybritz/reinforcement-learning” under the MIT-license, from the
Vizdoom environment. The algorithms used in the Vizdoom environment were REINFORCE, DDQN, Advantage Actor-Critic and Advantage Actor-Critic with LSTM. The Vizdoom algorithms were based on work by (Yu, 2017) under the MIT-license, who implemented these algorithms mentioned above with Keras layers as neural network function approximators. The implementation was changed to accommodate for richer reward signals, as well as optimized for quicker training by removing renderings and sound from the training GUI. The source code used for the agents can be found at the GitHub “https://github.com/flyyufelix/VizDoom-Keras-RL” from the “ddqn.py”, “a2c.py”, “a2c_lstm” and “reinforce.py” GitHub files. In each of the files the shape of the reward of the agents is changed to include “AMMO” and “HEALTH”, as well as changing the rendering of the graphical user interface to make for shorter training times. The audio output of the interface was also deactivated to make the training faster. Information on how to make these changes is described in the Vizdoom documentation (Vizdoom team, 2018) The training was made on five standard computers each with one algorithm running, and the computers ran each for 48–62 hours. By the end of training the algorithms have trained up to 7,000–21,000 episodes depending on how quick the algorithm was able to train.

Results

Generally, the algorithms performed similarly between the two different environments. This means that the Critic part performed relatively better than the Actor part algorithm in both environments, and the hypothesis that increasing complexity would change the performance between the actor and the critic part could not be proven. The performance of the algorithms is summarized by each environment below.
Results by the environments

Cliff walking

Cliffwalking comparison - Q-learning, Policy Gradient and Actor-Critic

Figure 7. In the chart above the performance of the algorithms that were tested in the cliff walking environment, Q-learning shown in blue, Policy Gradient shown in orange and Actor-Critic shown in yellow can be seen. The X-axis is the number of trained episodes. The Y-axis is the performance and the amount of reward for one episode. The reward of -12 is the maximum reward for the cliff walking environment.

Cliff walking. It can be noted from figure 7 above that the Actor Critic starts out with negative performance values directly in beginning of training, but quickly improves its performance and finds an effective optimal strategy. Q-learning is showing effective results in performance compared with the other algorithms. The policy gradient algorithm used, showed a comparatively low performance compared to the other algorithms. All algorithms seem to converge approximately after around 100 trained episodes. The Q-learning algorithm gains performance comparatively the most rapidly in early training.

Vizdoom environment. In the Vizdoom environment, DDQN(Double Deep Q-learning) seem to be quickest at increasing its performance as seen per episode. Double Deep Q-learning is a slightly more advanced version of the basic Q-learning algorithm, that was used in the cliff walking environment. The Double Deep Q-learning had a relatively higher performance earlier in training compared to the other algorithms. In later training the performance of the Actor-Critic was at a higher relative performance. The Actor-Critic had on the other hand a much higher observable relative variance in performance compared to the other algorithms. An Actor-Critic version with LSTM (Long short-term memory) boost had even higher observed relative variance, but on the other hand it seems to perform at a near perfect performance at the end after 16,000 episodes.

The Policy Gradient algorithm REINFORCE had a low relative performance compared to the other algorithms. Since it is a more complex environment it takes more
episodes of training for the algorithms to reach convergence in performance compared to the cliff walking environment.

While the DDQN was the most efficient at increasing performance per episode, the DDQN also took longer time to train per episode than the other algorithms. The DDQN algorithm had at the end of training been able to train for half the number of episodes compared to the other algorithms. Two figures are shown below (figure 8 & 9) illustrating the performance. Figure 8 shows the first 6,000 episodes where there is data for all the three major algorithms tested, while figure 9 shows data for the whole training duration for all the algorithms, but without data for the DDQN algorithm for the later parts of training. The Actor-Critic algorithms used was also tested with a version that was boosted with a LSTM network, which keeps better track of sequential events when training the function approximator. It can be considered unfair to include the Actor-Critic with LSTM in the main comparison between the algorithms, because the other types of algorithms were not tested with a LSTM boost.

![Vizdoom comparison - A2C, DDQN and Reinforce](image)

*Figure 8.* The chart above shows the performance of the three algorithms Advantage Actor-Critic (A2C), Double Deep Q-learning (DDQN) and REINFORCE. The X-Axis is the number of episodes trained. The Y-Axis is an indicator of performance, in this case the monster kill count where 25 is the maximum. The DDQN seems to be quick to increase its performance, and converges at a high performance. The Actor-Critic (A2C) seem to perform at higher maximum values than DDQN, but the Actor-Critic algorithm also seems to have an increased observed relative variance compared to the DDQN algorithm.
Figure 9. This is the same trial as shown in figure 8, but shown for the whole duration of all episodes trained. The chart shows the performance of four algorithms: Actor Critic (A2C), Double Deep Q-learning (DDQN), REINFORCE and Actor Critic (A2C) with LSTM (Long short-term memory) boost. The X-Axis is the number of episodes trained. The Y-Axis is an indicator of performance, in this case the monster kill count where 25 is the maximum. Note that the DDQN algorithm was slower in training and no data after 6000 episodes was obtained. The A2C algorithm with LSTM shows remarkable observed relative variance and near perfect performance at the end of training.

Discussion

As we can see from the results, the hypothesis, which was to find a difference between the actor and critic part when changing the complexity level of a task, could not be verified.

This could be because technical problems in the implementation, such as that the algorithms in each of the environments are implemented somewhat differently, and this difference in implementation could affect the results. Other technical issues that could influence the results, could be a difference in performance efficiency between the algorithms. The “REINFORCE” algorithm is an old and comparatively unrefined algorithm, and is not the state of the art for policy gradient algorithms anymore (Sutton and Barto, 2017), while on the other hand the algorithms used for the Critic part “Q-learning” and “Double Q-learning” is state of the art. This could be like comparing apples and oranges, and for future studies it would be suitable to use algorithms with more similar level of refinement and performance efficiency.

It might also be that the environments are so different that a comparison between the environments does not say anything about the difference in the algorithms performance. In this current experiment it is hard to compare the complexity between the environments with each other in a standardized way. It is even harder to compare how the complexities of the actions space and state value space change respectively.

To produce a better comparison between the algorithms it would be suitable to use only one environment and change the complexity variables of the environment and action space in a controlled manner. If you would use an environment with variables defining its complexity, then the complexity could be changed incrementally, and it would be possible to prove that a certain change in relative performance between the algorithms, was because of a given change in complexity that was just made. With more environmental control it would be
possible to only use one testing environment instead of several, which would give more accuracy to the comparisons.

It could matter if the complexity structure increases uniformly or not, and if one would assume that the problem structure had something to do with which part of the Actor-Critic algorithm that would be used most efficiently. Then there is the dimensionality in both the action- and state space to consider. It could be reasonable to think that finding the best working algorithm part, is to find the part that optimizes the best with the problem in the task or action space, which is the part that has the most covariance with the target reward function. If either of these spaces have a high dimensionality and suffering from a low degree of covariance with the target reward function, there would be a risk of overfitting the algorithms with poor results in training. A high dimensionality can in this case be interpreted as being interchangeable with high complexity.

If the complexity is uniformly increased between the environments in both the action and value dimensions, and like in this experiment, the Critic part and Actor part had the same relative performance in both environments, this would suggest that the increasing of the complexity of a problem by itself does not influence a dual processing’s relative performance. Then this notion could support the idea that there are two distinct types of dual processes that each can solve problems in a different way and is complementary with each other. But while these results might hint that is the case, to conclude if this is true or not, would require more studies with more detail and control over how each of the complexities of the action- and value spaces changes respectively. Further studies could also shine a light on whether other areas of cognition, such as psychology, could be benefiting from using the models of the actor- and critic estimations, as discussed next.

**Mapping the algorithms to cognition**

There have been multiple studies that suggests that is possible that the brain of mammals is having features similar to an Actor-Critic algorithm (Niv 2009). If the human cognition works in a similar way as the dual processing theory Kahneman (2011), and Evans (2008) suggests, it could be possible to reverse engineer the parts of the Actor-Critic algorithm theory, back to the properties of human cognition as proposed by Evans (2008) and Kahneman (2011). It seems that the Actor-part is similar to Kahneman's system 2, which is deliberate and provides rational ideas for modes of action. On the other hand, the Critic-part seems to be similar with Kahneman's system 1, which provides more spontaneous feedback and imagination, is more unconscious, and is used for creative insights in problem-solving (Kahneman, 2011).

**Applying math to cognition:** If the assumptions above hold true then it could be possible to look at the math of Actor Critics and try to predict what that could mean for our behavior. The basis is that the optimal policy is evaluated as: $Optimal\ policy = Value\ of\ actions \times Value\ of\ states$. This means that if one either one of the systems is faulty, either the state value or action estimate, the whole estimate will be flawed due to the multiplication that affects the whole product. If there would have been addition instead it wouldn't have mattered much if the estimate in one the systems was wrong, but with multiplication both estimates have a big effect on the end result, without being able to compensate much for the other systems possible fault.
This comparison is made on a higher order executive function schema level, and not at the neurological level as other authors have previously covered (Niv 2009).

**Mapping in psychology:** When mapping dual processing theory system 2, to action related processes as the Actor-part, and the corresponding system 1 to the Critic-part, interesting predictions can be made. If you assume that our actions are calculated in one product and that the values of the states in another product. This would mean that the systems are interdependent on each other’s accuracy, and that overcompensating for a faulty subsystem with another well-functioning subsystem will not work.

There are two dominant therapy treatment paradigms in psychology, one is more focused on immediate actions and is named cognitive behavior therapy(CBT), while the other is more concerned with imagination, ideas, and values and is named psychodynamic insight therapy(PIT) (Leichsenring et al., 2015). Since cognitive behavior therapy generally is concerned with immediate actions, and psychodynamic insight therapy is concerned with more abstract values and ideas, it would be logical to assume that the cognitive behavior therapy is primarily affecting the system 2/Actor part, while psychodynamic insight therapy is affecting the system 1/Critic part.

**Predicting treatment success:** If the assertions in the sections above hold true an interesting prediction could be made for the treatment successes of the two main psychological therapy methods CBT and PIT. If we assume that disorders that concern immediate actions, such as for example obsessive compulsive disorder(OCD), would be best treated with a therapy that is closely aligned with actions such as CBT. While disorders with less concern to immediate action and instead more concerned with values and global states of minds, such as personality disorders would best be treated by PIT. The prediction would then say that disorders that matches closely with a treatment at the end a spectrum would be untreatable with a treatment method from the other end of the spectrum, and vice versa. A quick literature search confirms that this could be the case, CBT is significantly effective for treating OCD while PIT is significantly ineffective for treating OCD (Leichsenring et al., 2015; Foa 2010). On the other hand, it does not seem to be the entire picture, because when looking on the other side of the theorized spectrum, does personality disorders seem to be treatable by both PIT and CBT (Leichsenring et al., 2015; Matusiewicz et al., 2011). These predictions are a bit out of the scope of the thesis, and builds upon quite general assumptions, but if the assumptions hold true, the potential usability in the field of psychology would make the idea worth considering.
Reference list


