Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Hegdal, Sondre Steinsland

A CBR-ANN hybrid for dynamic environments

Master's thesis in Master of Science in Informatics
Supervisor: Kofod-Petersen, Anders
February 2020

NTNU
Norwegian University of Science and Technology

Hegdal, Sondre Steinsland

A CBR-ANN hybrid for dynamic environments

Master's thesis in Master of Science in Informatics
Supervisor: Kofod-Petersen, Anders
February 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis presents a description of how to let an AI agent learn to adapt to many different situations in a dynamic environment. The method suggested is a Case-based reasoning and artificial neural network hybrid system, which makes the case-based reasoning system chose between several expert neural networks trained with reinforcement learning. The environment chosen to test the solution on is the video game Mega Man 2 for the Nintendo Entertainment System. The motivation behind the project is to expand on neural networks, as they are seen to be very good at learning a single task in an environment in previous works. However, they tend to struggle when the hypothesis space of the environment becomes too large, and might only learn a small subset of the total environment. Therefore it is suggested to combine the neural networks with case-based reasoning, allowing the system to choose between the most fitting experts for different subsets of the entire hypothesis space. The contributions made in this thesis is a new method for dynamic, continuous environments, that solves the problem better than the baseline tested against in this thesis. The thesis also offers a good base for further research, with its extensive future work section. The thesis follows a deductive research approach. It explores the related literature, before designing a test for the suggested CBR-ANN hybrid solution. The method is then be tested against a pure artificial neural network approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter includes a brief overview of the background and motivation for this project, as well as describing the goals and research questions for the project. The chapter also describes the research methods, and briefly the contributions of this project. Finally it describes the structure of the entire thesis.

1.1 Background and Motivation

This project is based in the artificial intelligence field withing computer science. Within this field it is in the machine learning discipline, focusing on dynamic problems, using artificial neural networks and case-based reasoning in combination.

The main motivation for the project is to improve on how neural networks performs in dynamic, continuous environments with big hypothesis spaces. The suggested method builds on having neural networks as experts for a small domain of the environment, as can be seen in Bling [2015]. Here the neural networks trained through NEAT [Stanley and Miikkulainen, 2002], performs well at a single level (case), but does not perform well on other levels it has not trained on. To improve upon this, a case-based reasoning approach is suggested, trying to overcome the problem of having too large of a hypothesis space by extending the amount of experts from a single neural network solution, to one neural network for each case the case-based reasoning system discovers. Bling [2015] was also a motivator for using games as the testing environment, together with the work done by google's Deepmind AI, playing Go Silver et al. [2017], Atari Mnih et al. [2013] and Starcraft II Vinyals et al. [2019].

1.2 Goals and Research Questions

Goal Explore whether the suggested CBR-ANN hybrid is a suitable solution for working in dynamic environments with large hypothesis spaces.

The goal is broader than the scope of the thesis, but will be answered in part through the following research questions. As will be seen in the structured literature review in Chapter 2, the suggested method is not very well explored and therefore hard to answer through purely reading previous research. To reach this goal, it is therefore decided that a system needs to be designed around the suggested method, and will be tested against a method that has previously been used for dynamic problems. If it is a suitable option or not can then be decided based on how it performs compared to other methods. The method of evaluation is described in Chapter 3. Evaluation criteria will be referred to as both evaluation criteria and reward throughout the thesis.

Research question 1 How will the suggested CBR-ANN hybrid perform compared to a single neural network in a dynamic, changing environment?

The research question is meant to answer a part of the goal presented above. It sets up the new suggested hybrid method against an established method to see how well it performs in comparison. The comparison will be done on the game Mega man 2. This game was chosen because of how each level can be seen as a separate dynamic environment, with the changing variable of which level the agent is currently on. In addition to this, the level select system is made in such a way that it will be able to highlight the strength of the CBR-system in the suggested method. More on the game can be seen in Section 2.1. The comparison will be done with the evaluation criteria described in Chapter 3. Answering this research question will help understand the main goal better by seeing how well the system performs compared to one other method. Several additional research questions related to the literature review are presented in the structured literature review protocol.

1.3 Research Method

The research model of this thesis is split in two. It uses both a theoretical approach through the literature review, and an experiment to further gain answers to the main goal of the thesis. The main research question is answered through the experiment, while the research questions related to the structured literature review are answered with the theoretical approach. The theoretical approach is a suitable approach because the literature will help give answers towards the main goal, through other experiments and projects with the suggested method,

or similar methods previously. The experimental approach has been chosen as the most suitable to answer the main research question, because it will answer the main research question the best, regardless of what has been found or not in the literature on the suggested method.

1.4 Contributions

The contributions of this thesis are a potential new method that outperforms the baseline it was tested against, and a good basis for further research on the method, both for this problem and for adapting it to other methods. The contributions are expanded on in Chapter 5.

A shorter paper has been published based on the work done in this thesis. The paper was presented at the Case-Based Reasoning and Deep Learning workshop at the 2019 International Conference on Case-Based Reasoning [Hegdal and Kofod-Petersen, 2019]. It has some slight differences in the experiment, both in the setup and exact results, but overall contribution showed the same result as the thesis.

1.5 Thesis Structure

Chapter 2 presents all necessary information for reading the thesis, including details about the game being used in the experiment. Chapter 2 also includes the structured literature review. Chapter 3 presents the model that the thesis explores. Chapter 4 describes the experiment and the results of the experiment. Chapter 5 discusses and evaluates the results presented in chapter 4, how they tie into the goal and research question and how the thesis contributes to the field. Chapter 5 also includes an extensive future works section.

Chapter 2

Background Theory and Motivation

This chapter presents all necessary background information on the concepts used throughout the thesis, as well as the motivation and literature review. Section 2.1 presents the environment used for testing purposes in this thesis. Section 2.2 presents the game used for testing in detail, as well as presenting the theory needed for understanding the CBR and ANNs used in the thesis. Section 2.3 contains the structured literature review protocol followed while collecting data for the structured literature review. Section 2.4 describes the motivation for the project, and presents the structured literature review.

2.1 Dynamic environment

This section describes the environment chosen as the test environment. The game used is the American release of Mega man 2 released for the Nintendo Entertainment System (NES) in June 1989. This section describes the game with its rules and terminology used throughout the thesis, and presents why it is a suitable game for the proposed method explored in the thesis.

2.1.1 Introduction

A game is the ideal test bed for the proposed method, as it can be both dynamic and continuous, making it suitable for the goal and research question of the thesis. Within the different games to choose from, the Mega man franchise stands out as especially suited for the proposed CBR-ANN hybrid. This is because of its level-system, where you have 8 unique levels that can only be beaten once each,

followed by the last few levels which are beaten in a set order, leading to the final boss.

Each of the original 8 levels gives at least one unique upgrade to the player character, which can help in the other levels it did not beat yet. The first 8 levels can be done in any order, making it have potentially 8! different paths to defeat the final boss of the game. After beating a level, it is clearly marked on the level select screen.

Because of this level system, it is very naturally made into cases for a CBR-system, making it very suitable for testing this system. The natural cases to make here would be combinations of which level the agent is doing right now, and which levels it has already done.

Mega man 2 has been chosen above later games of the series because of its documentation regarding memory addresses needed for making a reward function, as well as for making the input to the neural network.

Compared to Mega man 1 there is not much difference. Both are well documented and follow the same structure, but Mega man 2 has been chosen because of previous experience with the game.

The American release was decided in favor of the Japanese release because of the added difficulty setting, making the agent able to play at a slightly easier difficulty, potentially letting it get further into the levels. The difficulty setting affects how much damage the player and enemies takes, reducing it for the player and increasing the damage taken by enemies.

There are 12 different powerups, also referred to as “suits”, the player can wear, having different powers to utilize. The standard suite you start with has a basic weapon that works against everything, while the eight suits you get for defeating the boss in each of the levels has some enemies they work well against, and some they don’t work well against. How they operate also differ from each other. Lastly there are three suits with tools to help you traverse the levels, these are also referred to as “items”.

The player can choose between any of these powers as long as it has been unlocked and the player do not have any active powers currently. This means that you can never use the power obtained by beating the level you are currently in.

A power is active after pressing the ”shoot” button, resulting in the effect of the suite being activated, most of the time in the form of a bullet. With the exception of the standard suite, all the suits have a certain amount of energy, which is reduced every time the suite is activated. When this energy is depleted, the suit will not be able to activate anymore.

The player interactions of the game is rather simple. You can move left and right, jump and shoot. In addition, you can climb up and down ladders. The goal of each level is always to the right of the starting point, and always ends in

a boss fight. The levels have enemies in them, which will take away hit points from the player if hit by them or their bullets.

If the player's health points get reduced to 0, or the player falls into a pit, the player will lose a life. If the player runs out of lives, the game is over.

The enemies can, when killed, spawn certain items to help the player: a big and small orb for refilling the player's health points, a big and small orb for refilling the energy of the active suit, and an extra life.

2.1.2 Levels

Even though all levels in the game have their goal to the right of where the player starts and ends with a boss fight, they all have their own unique gimmick to get there. A comparison can be seen in Table 2.1. As can be seen in the table, most of the levels follow the same general layout of going mostly to the right, with some exceptions, but the gimmicks makes the levels very different to traverse, and could potentially make it very hard for a single neural network to learn how to go through them all, considering the results by Bling [2015].

Especially Quick man and Crash man are very different compared to the other levels, as their general structure is altered compared to them. All the levels are divided into 255x255 pixel screens.

A screen transition can both be at the end of a section in the level, going right, up or down, or it can be going left or right on a longer section, without having a graphic transition. The levels have obstacles and enemies moving based on how the player moves, and certain obstacles that move on its own regardless of player interaction, as long as the player is within vision of it, making them dynamic environments.

The different enemies also have different amount of health, takes different amounts of damage from the various weapons, and behave different towards the player. The input format however does not take this into consideration, and makes all enemies look the same for the agent.

2.1.3 Powerups

As mentioned in the introduction to the game, there are 12 different suits. Nine of them are weapons and three are tools to navigate the levels. Of these nine weapons, all but the "Mega Buster" is locked when the game starts, and are unlocked by defeating the appropriate level and its boss. As levels can not be entered again after defeating them, the weapon you gain from one boss can never be used against this boss.

The main difference between the weapons are how strong they are against the other bosses in the game, and how they can help make the fights a lot easier. They can also be used on the regular enemies in the levels.

The differences in damage done to the various bosses can be seen in Table 2.2. However, it isn't only in the damage that the suits differ from each other, they also differ in how the bullets work:

Mega Buster The player has this unlocked from the beginning. Fires small bullets in a straight line.

Bubble Lead Obtained after defeating Bubble man. Fires a bubble that follows the ground in the direction it was shot. If shot in the air or going outside a cliff, it will fall down.

Air Shooter Obtained after defeating Air man. Shoots three whirlwinds that go in the direction it was shot and up.

Quick Boomerang Obtained after defeating Quick man. Fires a bullet that goes a short distance before returning to the player's position.

Leaf Shield Obtained after defeating Wood man. Creates a shield around the player as long as it is standing still. If the player is moving, the shield will be shot in that direction.

Crash Bomber Obtained after defeating Crash man. Fires a bomb in a straight line from the player's position that damages enemies and can stick to walls. It can break certain walls in Flash man and Heat man's levels.

Time Stopper Obtained after defeating Flash man. Stops movement of all enemies and moveable platforms in levels. Once fired it can not be canceled and the player can not change suit until it runs out of energy.

Metal Blade Obtained after defeating Metal man. Fires a blade in all of the 8 possible direction the player can press.

Atomic Fire Obtained by defeating Heat man. Fires the same way as the Mega Buster, but can be charged up for extra damage by holding the shoot button before releasing the shot.

Item-1 Obtained by defeating Heat man. Creates a platform that is slowly floating up, that the player can stand on to reach platforms it previously could not. While the platform is still active, the player can not change suit or place another platform.

Item-2 Obtained by defeating Air man. Creates a platform that travels horizontally, can be used to reach platforms that was previously unreachable, or to skip past hard platforming sections of a level. While the platform is still active, there can not be placed a new one, and the player can not change suit.

Item-3 Obtained by defeating Flash man. Creates a platform that sticks to walls and climbs them, and can be used to reach otherwise unreachable platforms. While the platform is active, a new one can not be placed, and the player can not change suit.

Table 2.2: Comparison between weapons on how good they are against the different bosses. Numbers are collected from strategywiki.org [2017]

Powerup	**Bubble man**	**Air man**	**Quick man**	**Wood man**	**Crash man**	**Flash man**	**Metal man**	**Heat man**
Mega Buster	1	2	2	1	1	2	1	2
Bubble Lead	-	0	0	0	1	2	0	5
Air Shooter	0	-	2	2	7	0	0	2
Quick Boomerang	2	2	-	0	1	0	4	2
Leaf Shield	0	10	0	0	0	0	0	0
Crash Bomber	0	0	4	2	-	3	0	0
Time Stopper	0	0	14	0	0	-	0	0
Metal Blade	4	0	0	2	0	4	-	1
Atomic Fire	0	2-6	2-10	1-14	1-6	2-6	1-4	-

2.2 Background Theory

This section introduces the necessary background theory and information used throughout the thesis.

2.2.1 Case-based reasoning

Case-based reasoning is a system that bases itself on saving experiences as cases [Mitcher and Weber, 2013]. A case in this context is a set of features that you can distinguish from each other. A case-base is a collection of these cases[Mitcher and Weber, 2013].

It uses reasoning techniques to retrieve the cases stored in a case-base, and alter them to fit new problems. The new solutions to the new cases can then be saved as a new solution to a new experience [Mitcher and Weber, 2013].

Maybe the most important part of understanding CBR is the CBR-cycle. It has the four steps or phases: retrieve, reuse, revise and retain [Aamodt and Plaza, 1994].

This subsection will explain in order the cases and their representation, similarity measures and the steps in the CBR-cycle and how they relate to each other.

Case representation

Cases in a CBR-system refers to an instance or experience that consist of a problem and a solution [Mitcher and Weber, 2013]. A case is represented through attribute-value pairs [Mitcher and Weber, 2013]. This is used to represent the state of an entity, which in the case of the experiment performed in this thesis is e.g. which the level that the agent is currently playing. The attribute in this case would be level, while the value would be which level the agent is currently in. This example is explained in detail in Chapter 3.

An attribute can both contain a single value or a collection of values [Mitcher and Weber, 2013]. A case can consist of several attributes that describes it [Mitcher and Weber, 2013].

Similarity measure

The similarity measure in a CBR-system is how you compare the different cases based on their representation [Mitcher and Weber, 2013]. This is done by comparing how similar each attribute in a case is, and by evaluating how important each of these attributes are. The simplest way of representing this is to use a table which has all the similarities between the different values an attribute can have.

There are two terms that are important for the similarity measure: local similarity and global similarity. Local similarity compares the values within a single attribute to determine how similar this attribute is between the different cases [Mitcher and Weber, 2013]. Global similarity determines how similar cases are overall, by looking at the similarity of all the different attributes the case can have, while using the importance of each attribute to reach a final conclusion on the similarity [Mitcher and Weber, 2013].

The simplest way to represent the importance of a case is by having the local similarity be a value between 0 and 1, and then having the importance represented as a multiplier that increases with increased importance. The final similarity would be the sum of the similarities of each attribute multiplied by the

importance [Mitcher and Weber, 2013]. Importance can also be described as the weight of each attribute.

Retrieve

This is the first step of the CBR-cycle, and happens when the agent encounters a new problem [Aamodt and Plaza, 1994]. This is where the CBR-system searches through its case-base, which contains all of its previous experiences, to see if it has encountered something similar in the past [Aamodt and Plaza, 1994]. To find this, the system uses what is called a similarity function [Mitcher and Weber, 2013], to see how far apart the new problem is to the previous cases.

When the system finds a suitable case in its case-base, that solution gets selected to solve the new problem [Aamodt and Plaza, 1994].

The method used and described later in this paper for this is a weighted sum function. After it has retrieved the most similar case from memory, it continues to the reuse phase.

Reuse

The reuse phase takes the solution to the most similar previous case and tests whether this solves the new problem or not [Mitcher and Weber, 2013]. This is taken as a suggested solution for the case. Reuse has two possible ways it can use the case, either copy the existing solution or adapt the existing solution [Aamodt and Plaza, 1994].

This thesis uses the copy method, and lets the revise phase of the system take care of the adapting the solutions. After the reuse step is done, it is sent to the revise phase.

Revise

In the revise phase, a solution from the reuse step is evaluated, and changed if it is not considered correct [Aamodt and Plaza, 1994]. The system then takes this opportunity to learn from its experiences. If it is considered the correct solution, it does not need to get altered, and is sent to the retain phase, otherwise it is altered to fit the problem through using domain-specific knowledge or an expert [Aamodt and Plaza, 1994].

In this thesis it uses domain-specific knowledge through Q-learning and its reward function.

Retain

Retain is the last phase of the CBR-cycle [Aamodt and Plaza, 1994]. This phase looks into what to keep and what not to keep from what it learned in the previous

steps, and how to remember this information [Aamodt and Plaza, 1994].

In the case of this thesis, it saves the case as it was represented when presented to the system, and saves the new, altered neural network as the solution.

2.2.2 Artificial Neural Network

Artificial Neural Networks is a learning method that performs well when it comes to predictions of values with noisy inputs [Mitchell, 1997], this makes them ideal for controlling the player in dynamic environments. This subsection provides a brief overview of neural networks, including the recurrent neural networks used in this thesis.

It first describes the general structure of neural networks, before going into the forward pass used during both testing and inference and the backward pass that is used after a forward pass during training. It ends with a description of recurrent neural networks.

Structure

The structure of a neural network consist of an input layer, an output layer, bias nodes, hidden layers, neurons/nodes and weights [Mitchell, 1997].

The input layer is the data that the ANN can observe, and is what it has to predict a value for [Mitchell, 1997], e.g. be an image.

The output layer is the predicted value(s) that the network considers to be the answer for a given input, e.g. a vector of actions an agent can take, with the estimated reward for taking each action.

Bias nodes are nodes that have a set value at all times, and never changes [Mitchell, 1997].

Hidden layers are extra layers between the input and output layers [Mitchell, 1997].

Neurons/nodes are what the layers are built up from [Mitchell, 1997]. Weights are what connects the layers/nodes to each other, as well as connect layers with the bias nodes [Mitchell, 1997]. An illustration of this can be seen in Figure 2.1.

Forward pass

A forward pass consist of using the values of the input layer to go through the network iteratively and calculate the output of the neural network [Nielsen, 2015]. To do this, all the nodes in the input layer gets assigned a value corresponding to that of the input to the network. Each node in the next layer then calculates it's own value based on the weights coming from the nodes it is connected to, both from the previous layer and the bias nodes if any. This value is the sum

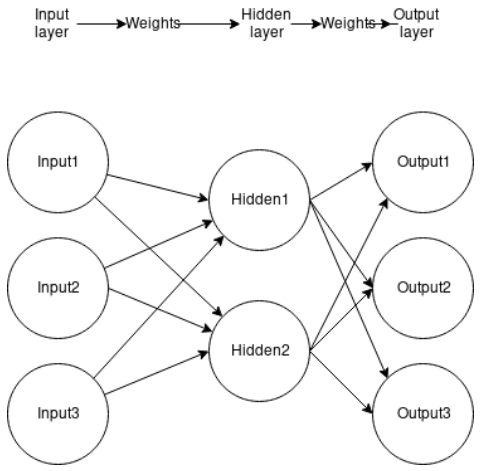


Figure 2.1: Example of a simple Neural network structure.

of all weights multiplied by its corresponding node-value. This value then goes through an activation function or transfer function.

Common choices are the sigmoid and tanh functions [Mitchell, 1997], as well as ReLU [Nair and Hinton, 2010] and ELU [Clevert et al., 2015]. This is done to ensure that the network can adapt to more than just linear problems [Mitchell, 1997].

The results in the final layer, following this method, is the output of the network. This is then used to decide what action or classification the network predicts as the correct answer. Often this is done through selecting the highest value of the output nodes [Mitchell, 1997] or using a softmax function [Goodfellow et al., 2016].

In this project it has been decided that the highest predicted value will be selected from the neural network outputs. This is because of how deep Q-learning functions. As Q-learning tries to learn to predict the possible rewards with each action taken, including future actions [Mnih et al., 2013], it makes it hard to use normalized values like softmax functions return. Normalized values in a system with unknown maximum and minimum values are hard to make reliable. The system only updates the network on the action taken [Mnih et al., 2013], softmax would not be ideal, as this would need to alter all outputs because they have to sum up to 1. Deep Q-learning will be expanded upon in Section 2.2.3.

During training, the forward pass will be followed by a backward pass.

Backward pass

A backwards pass or back propagation is what is done when you propagate backwards through the network, using the outputs from a forward pass [Nielsen, 2015].

This is done to update the weights if the result of the forward pass did not reach the correct conclusion, making the network learn how to solve a problem [Nielsen, 2015]. A common way of doing this would be to compare the result with the correct solution, and use the Mean Squared Error (MSE) to correct the weights [Mitchell, 1997].

Stochastic Gradient Descent (SGD) is commonly used for updating weights [Mitchell, 1997]. This method uses the error from the targets against the actual output, as well as the derivative of the activation/transfer function to find a delta of which the weights needs to change to get the correct answer [Mitchell, 1997]. This is then subtracted from the current weights, after being multiplied by a learning rate [Mitchell, 1997].

The learning rate is a value between 0 and 1, indicating how much the weights should change in each step, with 0 being no change and 1 being to change it to the perfect values for that exact output at every step. Usually this value is rather small, to not have the network forget what it learned previously in favor of learning the most recent case at every training example.

Recurrent Neural Network

Recurrent Neural Networks, is a special architecture or topology for a neural network. Instead of the regular feed forward networks that was shown above, the recurrent networks has weights that point backwards towards other nodes [Mitchell, 1997], creating something similar to a bias node.

This node, however, gets updated at every time step, using the output of the previous time step as its input, making the network able to remember inputs and outputs it has seen before Mitchell [1997]. This process can be repeated as many times as desired for remembering further back in time.

It has been decided a simple recurrent model will be used in this project in favour of using a more advanced model like an LSTM [Hochreiter and Schmidhuber, 1997]. This decision was made through experimenting with both, and finding that an LSTM would take too long to train, as it required more resources in terms of processing time. Using LSTM models are discussed further in Section 5.4.

An example of the architecture used in this project can be seen in Figure 2.2. This approach was taken as an evolution of how neural networks are used by Mnih et al. [2013]. Mnih et al. [2013] used regular feed forward neural networks, but used the state of the monitor in the last four time steps to generate the total input to the network. A recurrent neural network will be able to do this as well,

in addition to potentially remember details in the input and output that might get lost with the simple feed forward method [Mitchell, 1997]. Because recurrent neural networks works so similarly to the previous method used by Mnih et al. [2013] while also being able to remember more details of previous inputs [Mitchell, 1997], recurrent neural networks has been decided as the ideal approach.

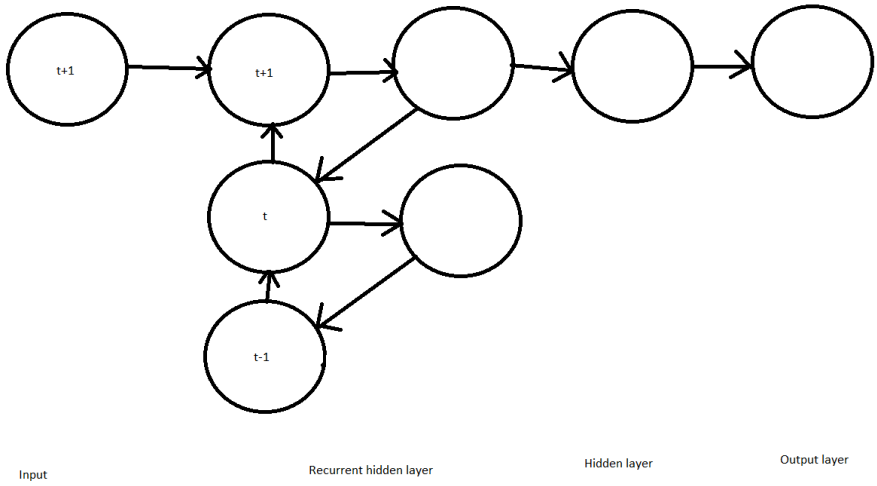


Figure 2.2: Illustrates the simple recurrent model used in this project. "t" refers to the time of the input, t+1 being the new state, t is the most recent input that is finished calculating and t-1 is the state before the current newest output.

2.2.3 Q-learning

Q-learning is a reinforcement learning technique, meaning it learns by getting rewards via trial and error, and refines its actions more as it gets better and better at predicting how much reward an action will give [Mitchell, 1997]. Because of its previous success in similar environments [Mnih et al., 2013, 2015; Wang et al., 2015], this approach has been used to train the neural networks in this project.

Basic Q-learning

Q-learning tries to learn the function:

$$Q(s,a) = r(s,a) + \gamma V * (\delta(s,a))$$

Where s is the state, a is the action taken, r(s, a) is the reward in state s, given action a, γ is the discount factor, meaning how much it should consider the rewards in the coming actions, and V*(δ(s, a) is the reward from taking the best path from this point on [Mitchell, 1997].

Using this function, the agent eventually learns how to act in an environment through trying and failing different actions [Mitchell, 1997]. This is can be done with a chance of a random action happening, so that the agent will learn the corresponding rewards for actions in states it has not seen before [Mitchell, 1997]. Eventually the agent will have built a good understanding of its environment, and can act well based on the rewards it expects to get through the different actions in different states.

The classic Q-learning approach is to have a table for all actions and states, that will get updated as the agent discovers the real reward of actions, and later actions in a chain of decisions [Mitchell, 1997]. The following function is used to update the table containing the expected Q:

$$\hat{Q}(s, a) = r + \gamma max_{a'} \hat{Q}(s', a')$$

Where $\hat{Q}$ is the current values for Q, s is the previous state, a is previous action, r is the current reward, γ is the discount factor, s' is the new state, a' is the next action and $max_{a'}$ gives a' the action that will give the best Q value, based on the new state and what the agent knows about the possible actions.

This approach is however not very well suited for a continuous game with large amounts of different states and actions, as the space needed to store all the states could outgrow the storage space and memory available. Therefore this project uses Deep Q-learning, which uses the same principles, but is altered to work well with training neural networks rather than use a table to look up the actions for a given state.

Deep Q-learning

Deep Q-learning (DQN) uses the same principles as the classic Q-learning algorithm, with maximizing Q. This method is however more suited for bigger domains than the classic Q-learning approach [Mnih et al., 2013].

DQN uses neural networks to estimate the rewards for each of the actions, and then updates the weights through backpropagation, using the difference between the predicted reward and the actual reward [Mnih et al., 2013].

The backpropagation algorithm then updates the weights in the network, rather than updating a table entry. As the actual next state and its reward is often unknown at the time of the prediction in the problems where DQN is utilized, the training needs to work around this [Mnih et al., 2013].

This can be done through having the network that is currently training also calculate the maximum reward for the new state that it discovers after an action

is done [Mnih et al., 2013]. This calculation of the next state replaces the table lookup in the classic Q-learning approach.

DQN as described above is the simplest approach, and it does works, but not without problems [Van Hasselt et al., 2016]. It is very easy for the agent to overestimate the reward it will get eventually, leading it to e.g. only executing a single action no matter the situation [Van Hasselt et al., 2016]. There are two main methods of fighting this common problem.

The simplest is double DQN [Van Hasselt et al., 2016], where the network that estimates the max Q-value for the next state is updated to become the main network at slower intervals rather than using the exact same network for both at every update. This makes the overestimating a much smaller issue.

The other method is dueling DQN [Wang et al., 2015], which uses a network that splits in two, one that estimates what they call Value, and one that estimates Advantage. Both of these output Q-values for each action, but they are handled differently in each one. They are then combined to a single output in the end.

This architecture is the strongest of them, both learning faster and more consistently than the others, though the difference between double DQN and dueling DQN is not massive. Double DQN is being used in this project, because of it being simpler to implement and both being able to improve the overestimation problem [Van Hasselt et al., 2016; Wang et al., 2015].

2.3 Structured Literature Review Protocol

To reach the goal and answer the research question presented in Chapter 1, a structured literature review is regarded as a suitable approach to learn what has previously been researched on similar topics. This will help get a grasp of how the state of the art performs with similar approaches and/or problems, while also getting closer to reaching the main goal of the thesis, which is to explore how the proposed CBR-ANN hybrid would be suitable for dynamic environments with large hypothesis spaces.

The structured literature review protocol has been created following the work presented in Kofod-Petersen [2012], and has its own set of research questions that it will answer. These questions will all help in furthering the understanding of the main goal, and can help in answering the main research question if any similar methods appear in the literature search.

The structured literature review will answer the following questions:

RQ2 What are existing solutions to solve problems with dynamic environments?

RQ3 How is CBR and neural networks used together today?

RQ4 How do the solutions in RQ2 compare to the proposed method of CBR and neural networks together?

RQ5 How strong is the evidence for the different solutions presented in RQ2?

RQ6 How will this affect the creation of the CBR and neural network system suggested?

2.3.1 Objective

The objective of the structured literature review is as mentioned earlier to answer the research questions presented, so that the thesis gets closer to answering the goal and main research question. In addition this, the structured literature review can give possible improvements for the method, based on what has been found in previous research, as well as other methods to test against and ideas for future work on the model, test environment and problem.

2.3.2 Evidence gathering and study selection

Search databases with predetermined strategy

- IEEE Xplore
- ISI web of knowledge
- ScienceDirect
- SpringerLink
- ACM digital library

The search terms and groups used can be seen in Table 2.3. The search terms are divided into groups depending on their theme.

The 5 groups are one for the environment, one for general machine learning and AI, one for case-based reasoning and one that is game related, as a game was chosen as the testing environment.

It will be searched with the constraints, each term inside a single group will always be an OR constraint, so only one of them has to be found to return a result:

1. Group 1 AND Group 2 AND Group 3 AND Group 4 AND Group 5
2. Group 1 AND Group 2 AND Group 3
3. Group 1 AND Group 2 AND Group 3 AND Group 4

4. Group 1 AND Group 2 AND Group 3 AND Group 5
5. Group 1 AND Group 2 AND Group 4 AND Group 5
6. Group 1 AND Group 2 AND Group 4
7. Group 1 AND Group 2 AND Group 5
8. Group 1 AND Group 2
9. Group 2 AND Group 3 AND Group 4

Reference search

Hand searched references of primary studies/literature that passed quality assessment.

Select primary studies

- Remove duplicates
- Remove same studies from different sources
- Filter studies based on date
- Only top 30 results are selected, sorted by relevance by the database searched, in cases of more than 30 results.

Study quality assessment

Filter literature on the list below:

IC1 The studies main concern is the problem of an agent learning to operate in a changing environment.

IC2 The study is a primary study presenting empirical results.

IC3 The study focuses on CBR.

IC4 The study focuses on ANNs.

IC5 The study describes a method to solve the problem of an agent learning to operate in a dynamic environment.

Where IC1 and IC2 are primary screening criteria, and IC3-IC5 are secondary.

QC1 There is a clear statement of the aim of the research.

QC2 It is put into context of previous studies.

QC3 Are the proposed methods justified.

QC4 Is the data set reproducible.

QC5 Is the method reproducible.

QC6 Is the experiment thoroughly explained and reproducible.

QC7 Is it clearly stated what it has been compared with.

QC8 Are the performance metrics explained and justified.

QC9 Are the test results thoroughly analysed.

QC10 Does the test evidence support the findings.

All QCs that return true will give 1 point, partly true will give 1/2 points and false gives 0 points. QC 8 and 9 are mandatory (1/2 or higher). To pass the quality assessment it is required to have at least 6 of the total of 10 points.

Data extraction and screening

The following steps will be performed in order:

- Database search for literature as described. All results will be saved to an excel-sheet with all relevant data (author, publisher, date etc.), that will be reduced in the next steps.
- Removal of duplicates.
- Filter by date if necessary.
- Check eligibility through the quality assessment. This will be done in a three step process:

    1. Abstract inclusion criteria screening. This is going through the abstracts to assess whether the inclusion criteria is met or not. The abstract should at least indicate that IC 1 and 2 are being met in the study, and should include at least one of the three secondary criteria, IC3-5. Exception here for studies concerning RQ3 and Search criteria 9, this will only have to contain IC 2 of the primary and should have both IC3 and IC4.

2. Full text inclusion criteria screening. This is going through the entire text to look for the inclusion criteria. It should contain IC 1 and 2 and at least one of the secondary criteria. Exception here for studies concerning RQ3 and Search criteria 9, this will only have to contain IC 2 of the primary and should have both IC3 and IC4.
3. Full text quality screening. Go through the entire text to evaluate the quality of the study by answering the ten quality criteria defined earlier.

2.4 Motivation

The motivation behind the thesis is to learn whether you can improve current methods in dynamic environments that might change drastically compared to the current state of the art.

One such environment can be seen in the work done by Bling [2015], which plays the game Super Mario World, and learns through NEAT [Stanley and Miikkulainen, 2002].

This method performs very well on a single level, but when faced with multiple levels, and even multiple sections within some more advanced levels, the method starts to fail.

To combat this problem of having an environment like the levels in games, which can be seen as several smaller environments on their own, the CBR-layer on top of the neural networks was presented.

This CBR-layer was made so that the different levels/environments or part of the levels/environments can have individually learned patterns that might fit that situation better, and also not forget the previous experiences it had through trying to learn other patterns later on.

From reviewing the literature, it does not seem like the proposed method is explored much for any problem, and does therefore give motivation to find out whether this method could be an improvement over more established solutions.

The few similar methods found in the literature does however look promising. These similar approaches will be explored more in detail below, and does exhibit better accuracy compared to using CBR and ANN approaches by themselves [Pinzón et al., 2010; Liu et al., 2006]. It is will be interesting to see whether or not the proposed method continue the trend found in these similar approaches.

There were also other approaches to combining CBR and ANNs that were more common than the proposed method, like the work seen in Biswas et al. [2014], which uses neural networks to find the feature weights used during the retrieval process.

Overall there were not many CBR and ANN approaches found in the literature search, but there were other hybrids and non-hybrid methods for dynamic environments found that included either CBR or ANN. Some of these can be seen in the works done by Liao et al. [2012]; De Paz et al. [2012]; Schlessinger et al. [2006]; De Angelis et al. [2013].

This section contains the structured literature review that is presenting the finds for following the method presented in Section 2.3. It discusses further the topics presented above, and answers the structured literature review research questions in the order RQ3, how is CBR and ANN used together today? Followed by RQ2, RQ4 and RQ5, what are the existing solutions to solve problems with dynamic environments, how do these solutions compare to the proposed CBR-ANN hybrid and how strong are their evidence. These are all answered in the same subsection. The section ends with discussing RQ6, how will the findings affect the creation of the CBR-ANN hybrid that is suggested.

2.4.1 Proposed CBR-ANN hybrid

This section presents a brief overview of the suggested CBR-ANN hybrid. This is expanded upon in Chapter 3.

The suggested CBR-ANN hybrid consists of a CBR-system that uses different ANNs as solutions to the cases it encounters. Each case in the case-base contains its own ANN. The ANNs used are recurrent neural networks. Each network is trained to fit its case through a reinforcement learning technique called Deep Q-learning.

The cases in the environment presented in Section 2.1 are the levels in the game. While in a single level, the same network will be giving outputs and/or train continuously. The CBR-part of the system will only exchange the currently working network if the level/case changes. If the same level is revisited, the previously trained network for that level will be used and trained if training is still active.

2.4.2 How is CBR and ANN used together?

This subsection takes on RQ3, how CBR and ANN are is used together today, and discusses the most common problems CBR and ANN hybrids are used for together today, as well as the most common way of combining the two methods. It then describes some other use-cases for CBR-ANN hybrids and then highlights the cases of CBR-ANN hybrids that are the most similar to the proposed hybrid.

The most common problems found where a combination of CBR and ANN is used currently is for recognizing network attacks. One such approach is described by Herrero et al. [2013], where CBR and ANN is used together in a multiagent system.

The ANN's task is to generate projects of the network traffic, while the CBR system categorizes it. Their method is called RT-MOVIC-IDS.

Their method was not tested against existing solutions, but their tests showed good accuracy for both normal situations, and for anomalous situations.

Other papers that tried to solve similar problems with CBR and ANN hybrids are Pinzón et al. [2009] and Pinzón et al. [2010] which try to protect against SQL-injection attacks.

Security is however not the only problem combinations of CBR and ANNs has been used. Another example is for data mining, where Liu et al. [2006] proposed a method for combining other algorithm with CBR systems, including neural networks.

This system used ANNs for the retrieval phase of the CBR cycle, which is the most common way CBR and ANN were combined [Guo et al., 2011; Biswas et al., 2014; Ma et al., 2009].

It showed vast improvements over using the techniques by themselves, with their other method, neuro-fuzzy systems, being better than the combination with an ANN. This can also be seen in Ma et al. [2009] where it's used for designing motorcycles. It shows slight improvements over other techniques.

This method is potentially an improvement that can be made in the future for the proposed CBR-ANN hybrid, but will not be used in this project to cut down on implementation time and the added training time for training the neural network used in the retrieval phase. This will be further discussed in Section 5.4.

Another successful approach to this method was Guo et al. [2011], who used a SOM [Kohonen, 1982] to find the most similar cases in his retrieval part of a CBR system.

Biswas et al. [2014] made an attempt at a similar method, using a neural network to train the feature weights used by a CBR system during its retrieval phase, it did however not show very good results when tested.

Another more distinguished approach to the CBR-ANN hybrids can be found in the work done by Henriet et al. [2012], which evaluated potential victims of accidental exposure to radiation.

Their system deploys the normal CBR cycle, but if an expert tells the system that the case is too different or not suitable for the new problem, it will use an ANN to adapt previous cases to solve the new problem. This is then saved as a new case in the case-base.

It showed good accuracy in its tests, and they expect it to improve the accuracy compared to other methods when the case-base grows larger.

The combination seen in the work by Choy et al. [2003] uses a CBR-ANN hybrid to select potential suppliers from a supplier list. The list of suppliers that it can choose from is the CBR part of their system, while an ANN is trained to benchmark the potential of each supplier. The system outperforms both a

standalone CBR system and humans in the experiment conducted.

Some of papers returned by the literature search suggested similar approaches the CBR-ANN hybrid presented in this thesis. One such approach is Pinzón et al. [2009], also mentioned earlier, which tries to detect SQL-injection attacks.

This method by Pinzón et al. [2009] uses CBR and ANNs together by having neural networks trained on previous, similar cases stored for future use. It uses a mixture of ANNs for each case, and trains the ANNs for a new case on the results that previous, similar cases used.

This approach is very similar to the proposed method for this thesis. It does however lack in some aspects to answer the main research question, as it gives out a single yes/no/maybe for each new case it encounters, rather than produce a string of output values before it's decided that a new ANN/group of ANNs should take over.

It does also not train on examples as they appear, like the proposed method of this paper does with reinforcement learning.

It would be interesting to see how a similar approach to theirs with multiple ANNs working together and training on the outputs of similar cases could affect the proposed method for this paper, to avoid one solution learning something that might also be useful in other solutions.

More discussion on this can be seen in the future work Section 5.4. Their method does show good results, beating all the methods they compared with, but only slightly.

Pinzón et al. [2010] is also very similar to the previously mentioned paper. This paper uses SVN/ANN for classification in the reuse phase of a CBR system to combat SQL-injection attacks.

It is also very similar to the suggested method here, but it does still differentiate itself with having an expert evaluate the results after a classification has been made, and it only uses one classifier for classifying once, rather than having the system choose a network to do a sequence of classifications.

It is however very reassuring that this, like the previous one, shows improvement over other previously used methods for their problems.

The most similar approach to the suggested CBR-ANN hybrid discovered in the literature search is presented by Zehraoui et al. [2004], which uses a method called CASEP2.

This method uses an ANN called M-SOM-ART for classifying or predict a cluster of tasks. It shows good results in the experiment conducted, but still requires more testing.

It will be interesting to see whether this method perform as well in a more dynamic world, and using reinforcement learning with RNNs rather than their suggested M-SOM-ART. De Paz et al. [2012] does also use an approach very similar to the suggested CBR-ANN method for their dynamic problem, evaluation

of atmosphere-ocean interaction.

This is done with a hybrid of CBR, SVN and ANNs, having SVN and ANN do the classification work in the reuse phase, and later changed in the revise step/retain step if necessary.

It is meant for online training and usage. It was tested on the problem they set out to solve, and their system had a strong accuracy, which grew when the number of cases in the CBR-system increased, it was not however tested against anything, so their evidence could have been better.

It differentiates itself from the proposed CBR-ANN hybrid with having the case only output a single classification, rather than having continuous output from it until a new case is appears.

These findings gives several answers to how CBR and ANN is used together today. In terms of how CBR and ANNs are used together in hybrid solutions, with combining them in different ways, both similar to the suggested CBR-ANN hybrid [Pinzón et al., 2009, 2010; Zehraoui et al., 2004], as well as different approaches [Henriet et al., 2012; Choy et al., 2003; Guo et al., 2011; Biswas et al., 2014; Ma et al., 2009; Liu et al., 2006].

In addition, they were used on a range of problems, both with cases requesting a single answer [Pinzón et al., 2009] and requiring a string of outputs [Zehraoui et al., 2004].

Some of the findings were interesting for future research on the proposed CBR-ANN hybrid, but there were nothing of interest as for improving the method found while answering this research question, and not much could be said towards the main research question of the thesis.

The results shown in the literature is however promising for the experiment, and is pointing towards the CBR-ANN hybrid performing better than the baseline agent it is tested against.

2.4.3 What are existing solutions to solve dynamic environments, how do they compare to the proposed method and how strong are their evidence?

This subsection takes on what solutions are the most common approaches to dynamic problems according to the literature search (RQ2), evaluate the evidence for how good these methods area (RQ5) and compares them to the proposed hybrid solution (RQ4).

Solutions using neural networks or case-based reasoning has been prioritized when filtering papers to include in the structured literature review, as those will more closely relate to the suggested CBR-ANN hybrid.

First neural network solutions are discussed, followed up by case-based reasoning systems, ending with a discussion on other methods entirely.

Galway et al. [2008] takes on AI and machine learning used in video games, as the non-player characters. The most relevant methods mentioned are neural networks and reinforcement learning.

The focus of this article is to be more on how it is already used rather than how good they can get, and if it makes sense to use from a business standpoint.

The biggest problem for using machine learning for this is the consistency of the agents, where it might be cheaper to make a machine learning agent with supervised learning for some cases. An agent using more traditional AI-approaches, however, might perform just as well or better for a given environment, without having to use time for training.

It also argues that offline training is better for these kinds of agents rather than online training.

This is worth taking note of for the proposed hybrid solution, as it might be a good idea to do some offline supervised training for the "base" neural network used before the CBR system has seen any cases, so it doesn't start from knowing nothing when the reinforcement learning begins.

Huang et al. [2015] uses neural networks to detect anomaly behaviour in large traffic data, which is a dynamic environment. It was tested on over 150 000 traffic samples. It was not tested against any other methods. As far as comparing against the proposed CBR-ANN hybrid, neither the method nor the problem is very similar, outside of them both using neural networks. The evidence they used could be stronger. The results it showed was good based on the percent based metric they used, but when it's not compared to a control algorithm it is hard to say how good it does compared to other algorithms that could be used for the same problem.

Schlessinger et al. [2006] uses modular neural networks as the brain of visual agents. They show improvement in behaviour compared to the non-modular networks that they tested against. It uses a principle that is pretty much the same as the proposed CBR-ANN hybrid, but exchanges the CBR part with just another neural network. It has one big neural network with a connection to the smaller ones, called a mixture of experts. It was tested by controlling agents in a game world where they try to survive by eating positive resources and avoid negative resources. Trained with an evolutionary approach. Their approach means it can handle subtasks dynamically, but was tested in a static environment. The evidence could be stronger with comparing against more or other methods than just a non-modular approach, but does show how well it compares to a solution that does not go by the principle of having a mixture of experts. It outperforms the non-modular version by a lot.

Schrum and Miikkulainen [2014] also utilized modular neural networks to have a mixture of experts to play the game Ms. Pac-Man. It was tested in a few preset game modes, and compared to other approaches used for the same game

that they found in the literature. The modular neural networks beat the average score of every other approach to playing the game. This shows great results for having a mixture of experts solving an issue, which is promising for the proposed CBR-ANN hybrid. Their evidence are rather strong, given that they compared against all other approaches found for this task.

Staying in the video game world, Tong et al. [2011] used neural networks, evolving the weights with a genetic algorithm, to play a sub-version of the game Warcraft III. The environment is dynamic, with having their opponent randomized, so the neural network agent has to be able to make counters to many different types of opponents. It is not compared to anything, but it does consistently beat the opponents given to it. In terms of comparing against the CBR-ANN hybrid, it is not very similar, and the way the games are controlled are very different for a real-time strategy game like Warcraft III and platforming game like Mega man. The evidence is not particularly strong with this paper, as it wasn't compared to anything, so it doesn't have any benchmark on how well it does against anything but the randomized opponent it fights against.

Munoz et al. [2009] is using neural networks a racing simulator. It used data outputted by three different agents to train in a supervised fashion, a human agent, an agent evolved with NEAT [Stanley and Miikkulainen, 2002] and a hand coded agent. These were used in all combinations of two with each other.

The supervised agent does not show particularly strong results, but it does show something important. The network performs better if its goal is just to finish, rather than finish quickly, as it tends to easily get stuck in a local minima that eventually turns out to be bad if it's too focused on getting through the problem fast. The evidence they show is good, comparing it to methods that are far superior to their own in most ways (NEAT) and also having a human to compare against, and a baseline hard coded agent.

Bahar et al. [2012] use neural networks for fast path planning in a corridor search. The agent operates in dynamic, unseen environments. The processing time makes it suitable for real-time applications.

Compared to the proposed CBR-ANN hybrid, it tries to solve some of the same problems with finding a path through an unseen, dynamic environment.

The approach is, however, slightly different, as their agent is planning a path, while the actions of the neural networks in the CBR-ANN hybrid are more reactionary.

It was tested on a robot arm, and compared to another algorithm meant for the same kind of task.

Their neural network beat the other in time used for the task by a lot. Their evidence could, however, be improved, as it was tested on static problems while claiming to work in dynamic and unseen ones.

Rahim et al. [2014] uses a genetic algorithm to train neural networks for the

action selection method used in a robot that is working in unknown/dynamic environments.

It was tested by having it control a robot in an environment they set up, and tested against a few other methods.

It's interesting to see how well this will do, as this network tries to do the same as the individual networks in the proposed CBR-ANN hybrid. The difference being their system only consist of a single network, and they control a real life robot rather than a video game character.

The evidence shown is strong, as it's compared to a lot of other methods. Most of the methods do well in a lot of the test cases, but there are some harder ones where the control methods fail, while the suggested method either solves them or get closer to solving them.

Where Rahim et al. [2014] tried controlling a robot using neural networks, Urdiales et al. [2006] tries making a reactive navigation technique.

Their goal is to make a CBR-system that can adapt to any robot and any environment through observation and self experience.

They achieve their goal for the test problem, which is to drive a small robot between a few boxes, in different tracks, and is compared to a few other methods.

This is very interesting for the proposed CBR-ANN hybrid, as the problem it is trying to solve could be exchanged for the problem that Urdiales et al. [2006] is trying to solve. This is especially interesting for Section 5.4 where it's discussed how the system can decide by itself when it should make new cases and not.

The difference is just exchange the actions the robot can do with neural networks that decide the appropriate actions. It is therefore very interesting to see how well this does.

All the methods they tested managed to do the test-tracks that they set up for it, but the CBR-system was slightly more efficient compared to the rest, cutting the corners a little bit closer when turning.

The evidence could be stronger, as they claim it is supposed to be able to use any robot in any environment, but only test a single robot for a set of what is inherently the same problem in different difficulties.

Sycara [1990] argues that CBR is better fitted for dynamic problems than rule-based expert systems and other expert systems that discard the solution after they either succeeded or failed at solving a problem. While these other methods discard their previous experiences, CBR saves saves them for future use, including the failures, so they are not repeated and/or it has tools for fixing the failed solutions. It was tested on a negotiation problem.

It shares some similarities in its approach compared to the proposed CBR-ANN hybrid. It is a CBR-system that tries to solve a dynamic problem, having cases based on some parameters in the negotiation process to decide how to react. Otherwise it is not very similar, as it is meant more for planning than reacting.

The results are rather weak, as it is not compared properly with anything, and does not have a performance metric to be judged on, only explanations of which actions it took in the negotiation process.

Another CBR approach to dynamic problems was suggested by Reyes et al. [2015]. They use a CBR and constraint satisfaction problem (CSP)[Kumar, 1992] hybrid in a dynamic, online environment.

It uses CSP for the retrieve and reuse phases of the CBR cycle. This allows for the CBR system to get online expert knowledge, and is according to the authors one of the biggest points in this article.

It was tested on some problems that was going to measure its ability to reuse previous knowledge, its ability to solve problems twice and the last to show the benefit of maintenance. It was tested against a CBR system without the adaptive knowledge method they used with their CSP. Their system performed slightly better.

Except for the maintenance part and how it works to solve dynamic problems, it offers little of interest for the suggest CBR-ANN hybrid. Their evidence for it working is rather lackluster, as their tests are very loosely explained, and are not very rigorous. It could have also been tested against other and/or better methods for those problems.

A similar approach to the CBR-ANN hybrid approach is the one from Morozs et al. [2016], which uses CBR coupled with Q-learning for dynamic environments. The approach is very similar to the CBR-ANN, with having the CBR chose a Q-table that fits the current problem, instead of selecting a neural network and train it with deep Q-learning.

The results showed a consistent improvement over the methods tested against, which is promising for the results of the CBR-ANN hybrid. The evidence was properly presented and discussed, but could have used some other algorithms to compare against, as one of them was just pure Q-learning, and the other performed worse than only Q-learning.

Gonzalez et al. [2003] used an instance-based learning approach, based on ACT-R, called CogIBLT. It was used for a dynamic problem, water purification, where the task is to distribute all water in a chain of water tanks within a set deadline, with the tanks being able to receive water from outside sources without the participant's knowledge at any time. The distribution happens by opening and closing pumps in the chain.

It was tested against humans. For the most part, CogIBLT did not outperform humans, but came very close to human performance on the task.

Compared to the proposed CBR-ANN hybrid, the CBR part and instance-based learning does share some similarities, as CBR is an advanced form of instance-based learning [Mitchell, 1997]. However, there is not much else they have in common. The problems they try to solve are also not very similar for the

two, other than them being dynamic.

The evidence provided is very strong for this paper, with rigorous testing being done on several different scenarios, and having the test-cases described well, together with good discussion on the results.

Horswill [1995] argues for the use of a group of specialized agents to solve complex, dynamic problems rather than having one agent do all the work. The principles are tested on a play-problem, having a few dynamic environments for action selection tasks, and give the environments a full state-machine.

The goal of this paper is, however, more on transformational analysis, and not so much the tests, meaning there are no comparisons on how well it performed. The paper has a lot of similarities and the same ideas as what spawned the idea of the proposed CBR-ANN hybrid, with having a specialized agent for a task, the neural network, for a specific dynamic problem, the case.

The evidence showed is very strong, providing twelve lemma to argue the points made, with the proof laid out for each one of them.

A multiagent approach was suggested by Garcia-Pardo et al. [2010] for changing environments. The agents are trained with social reinforcement learning through interacting with other agents and the environment, using Q-learning.

It was tested on a made up game where the agents should build or consume certain products for survival, and then the agents compete with the others for survival.

Except using reinforcement learning, this method is not very close to the suggested method in this paper. A multiagent architecture might be a good idea to try to incorporate for the CBR-ANN hybrid some time in the future, but will be discussed more in the Section 5.4.

The findings shows good promise, but the evidence is not very strong on how good it is overall, as it is only compared against non-social agents, where the social agents survive for a lot longer.

Xu et al. [2017] proposes a modified version of NSGA-II [Deb et al., 2000] suitable for dynamic multi-objective environments. It was tested on a few benchmark problems, against other state of the art algorithms, also NSGA-II variants, and beat them all at those problems.

As far as comparing against the proposed method, it tries to solve some of the same problems. They are both dynamic and multi-objective, but the approach is different with it being an evolutionary algorithm against CBR-ANN. It could however be interesting for future work to have an evolutionary algorithm instead of/in addition to reinforcement learning when finding solutions to new problems using old data.

The evidence for their results are strong, as they beat other state of the art algorithms.

Hong and Prabhu [2004] suggests a distributed reinforcement learning control

approach for multi-objective, dynamic environments. It performs very well on the test they set up for their paper.

This is very promising for the CBR-ANN hybrid, as it will also use this approach, though on neural networks instead of the Semi Markov Decision Process using Q-learning that is proposed in their paper.

Their evidence is strong, testing it on problems that are relevant for what they set out to solve, and comparing against a lot of different algorithms.

Another slightly different paper from the rest is Heywood [2015], which discusses a classifier system for dynamic tasks, using evolutionary computation.

It does not show any results of their system, but rather discusses what might be good and what might be problems with the approach.

One of them, which is the main problem the suggested CBR-ANN hybrid is trying to solve, is that the classifier might be very good at one part of the environment it is set in, while not being as good in another.

The CBR-ANN hybrid tries to solve this by having several expert approaches saved, and then find the right one for the right situation.

The work presented by Asada and Uchibe [2001] is building a reinforcement learning approach to learn in a multiagent system called robocup, where a team of robots will be playing football.

Their agents know nothing before starting. It was tested on a few simple game settings.

As it uses reinforcement learning in a dynamic environment, it's very promising for the performance of the suggested CBR-ANN hybrid method.

Their results are only tested against the previous method they implemented, so it could be stronger, but the behaviour seen in the new agent was a lot better than the previous.

De Angelis et al. [2013] use mixed-integer linear programming (MILP) for home energy management with dynamic electrical and thermal constraints. It tries to give an optimal solution to both keeping energy consumption down while keeping the house heated at a comfortable place for the user.

It is tested on a real-world application of the problem, using a heat pump.

Compared to the problem presented for the CBR-ANN hybrid, neither the method nor the problem shares a lot of similarities, except them being used for dynamic problems.

During the test, the users were very happy with the results, and the computational time needed to for the algorithm was also good enough that it was a usable solution.

The evidence presented is strong, as it proved to do well for a real world problem, both for power consumption and the maintained temperature.

What the existing solutions to solve problems in dynamic environments are got a lot of answers, especially Neural Network focused, which is promising as

neural networks does the bulk of the job in the proposed CBR-ANN hybrid.

As for how these findings compare to the proposed CBR-ANN hybrid, most of the methods does not compare well with the approach presented, and therefore does not help much in answering the main research goal either.

However, Morozs et al. [2016] does stand out in the literature found, as the proposed CBR-ANN hybrid uses a very similar approach. Instead of having a Q-learning table as the solution to a case though, the suggested CBR-ANN hybrid uses an ANN trained with deep Q-learning, as described in Section 2.2.3.

Tying this in with the main research question and goal, this does indicate that the proposed CBR-ANN method is a suitable approach to dynamic environments.

However, as the evidence was rather weak, the experiment set up in this thesis can still reinforce the results found by Morozs et al. [2016].

How strong is the evidence for the different solutions found can, in most cases found in the literature, be described as strong, having their methods either proven through argumentation or evaluated against previous state of the art algorithms.

There are however some that either did not compare their findings against other methods, or had a rather weak selection of comparisons.

2.4.4 How will this affect the creation of the CBR and neural network system suggest?

The work found does not offer much in designing the first iteration of the proposed method, as there is very little information to go on with this kind of system.

The results do however show some directions the system can take in future iterations, like maintenance [Lu et al., 2016] and in the retrieve phase of the CBR system [Liu et al., 2006].

For the neural network approach there was even less to work with in terms of design choices for the proposed method, as there wasn't much said on RNNs or DQN used on similar problems in the literature returned from the literature search.

One approach that could be taken from the papers would be how to train the networks with other methods in future iterations of the system.

Pairing ANNs with CBR systems for maintenance has been one of the methods to take from the literature review and into future iterations of the system, as it has shown good results.

One such paper is Lu et al. [2016] which shows the boost in performance the system gets by having good maintenance algorithms so the CBR part of the system doesn't get too cluttered with noisy data.

This will be especially useful for some of the methods discussed in Section 5.4, where the case-base is not as fixed as in the proposed first iteration of the system.

To answer how this will affect the creation of the CBR and Neural Network system suggested, the findings in the structured literature review does not impact the first iteration of the system that is presented in Chapter 3.

It does however give a lot of data to work with for suggestions in Section 5.4 where methods for future research on the suggested CBR-ANN hybrid solution, as well as other similar approaches to dynamic environments.

In the following chapter, the exact model that is used for the experiments in this thesis will be presented in detail, using the background information and findings in the structured literature review of this chapter as a foundation.

Table 2.1: Comparison of the different levels in the game, explaining their general layout and what the gimmick is

Level	**Layout**	**Gimmick**
Bubble man	A mostly going right level with some falling down occasionally	Falling platforms and being underwater, making all jumps higher and falling slower. Has instant death spikes.
Air man	Very similar layout to bubble man with mostly going right and some falling down	Platforms appearing when getting closer. Platforms can have enemies or obstacles that can hurt the player that has to either be killed or wait for them to go away, and moving platforms.
Quick man	Mostly falling down with some going to the right	Has lasers that will instantly kill the player if going too slow
Wood man	The simplest level layout with mostly going right and some down. Does not have any difficult jumps or enemies	Ladders are used for going down in the level
Crash man	Mostly climbing up ladders before having to go right	Climbing ladders throughout the stage and moving platforms on a rail.
Flash man	Mostly going right and down. Very maze-like level compared to the rest	Has ice physics, making moving around harder
Metal man	Mostly going right with some down	Has conveyor belts making the player move to the right and left by itself, and making the player faster/slower when moving on them.
Heat man	Mostly going right with some down	Has platforms that disappear and reappear at set intervals.

Table 2.3: Search terms

	Group 1	Group 2	Group 3	Group 4	Group 5
Term 1	Multi objective	Machine learning	CBR	ANN	Platfomer
Term 2	changing environment	AI	Case based reasoning	Artificial Neural Network	Nintendo
Term 3		Artificial Intelligence		RNN	Megaman
Term 4		supervised learning		Recurrent Neural Network	Super mario
Term 5		unsupervised learning		CNN	level
Term 6		reinforcement learning		Convolutional neural network	track
Term 7					map
Term 8					board

Chapter 3

Architecture/Model

This chapter contains the architecture of the proposed CBR-ANN hybrid that is explored in this thesis. It will include the overall architecture of the system, the specifics of the CBR-part and the topology of the neural networks used.

The tools used can be found in Appendix A, and the code is available online [1].

This section will contain information regarding the environment the experiment will take place in, as it is important for the representation and similarity measures in the CBR-system, the input and output for the ANN-part of the system, and the reward function used for the DQN.

Further information on the environment can be found in section 2.1.

3.1 Overall architecture

The system consists of one CBR part and one Neural Network part. It takes advantage of both neural networks' possibility of becoming experts within an environment [Bling, 2015; Mnih et al., 2013], and CBR's ability to use experts from previous experiences to solve new and unseen problems by reusing and alter previous experiences [Mitcher and Weber, 2013].

During training, the entire CBR cycle will be used. It will first retrieve similar cases from the case base, then reuse the network from the most similar case as the base for training a new network for the new case.

For revising, the network goes through a reinforcement learning process, using deep Q-learning to adjust to the new problem, and finally it will retain the case by saving the new modified network as the solution for the case.

[1] `https://github.com/imaltont/MasterCode`

When it's done with the training process and is using inference only, the two last processes of the cycle will be abandoned. New cases seen while the agent is not training, will only retrieve the most similar case it has seen before and reuse this, but not alter or save the solution for future use.

3.2 CBR

This section describes how the CBR section of the hybrid works. It goes through how the cases are represented and how their similarity is calculated.

3.2.1 Case representation

The cases are represented through two attributes with a custom value. The first is called "Level" and the second called "Powerup". The similarities between the different legal values in the two are represented through a lookup table, which is used for the similarity measure in the retrieve part of the CBR-system. These attributes are expanded upon in Subsection 3.2.2. Both attributes can have the following values:

- Bubble
- Air
- Quick
- Wood
- Crash
- Flash
- Metal
- Heat
- None (only in the "Powerup" attribute)

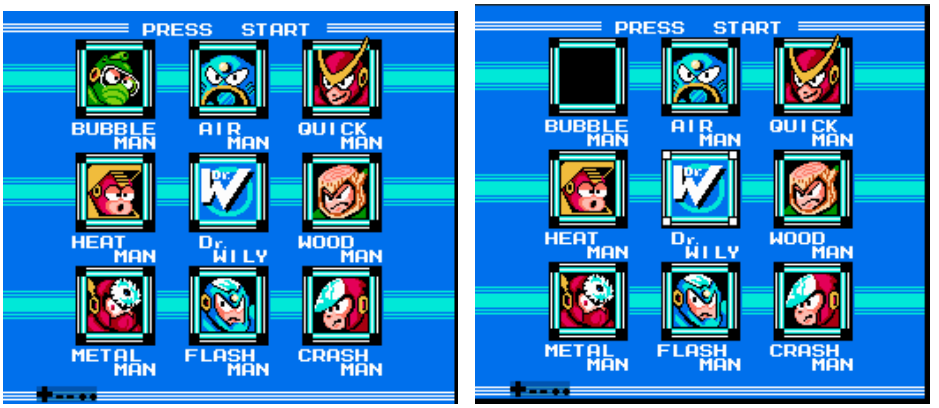
The "Level" attribute represents the current level that the agent is playing, while "Powerup" represents the levels that has been beaten previously in the current case, and tells the player which powerups are available.

This was chosen as the attributes to identify cases because it is information easily available to the player, as can be seen in Figure 3.1.

The levels that have already been done is marked with a black square, while the levels that are still playable shows the face of the boss of the level. The

order of the available levels are decided by the player, as described in Section 2.1, making the level a natural attribute to put into the case.

Only the "None" value was used for the experiments in the "Powerup" attribute because of time constraints, which will be expanded on in Chapter 5.



(a) (b)

Figure 3.1: The level select screen of Mega man 2. (a) shows the screen with no levels beaten, (b) shows it with Bubble man defeated.

3.2.2 Similarity measures

The similarity measure used in the retrieve phase is rather simple. It was decided to use a simple weighted sum approach when comparing the two previously mentioned attributes. This value is then normalized to be a value between 0 and 1.

If two cases has the same similarity, the one with the highest obtained reward will be chosen. The weight for both attributes are 1.

It is a simple approach to a similarity measure because the task is not to test the similarity measure, and therefore time can be more useful in other areas more relevant to answer the research question. Improvements to the similarity measure is a topic that will be discussed in Section 5.4.

In the following subsections, the similarity functions for each attribute is presented.

Level

The similarity measure for the "Level" attribute is decided by how similar the overall structure of the level is, which has been described in Table 2.1. The exact numbers used are between 0 and 1, and are based on how similar the author perceived them to be based on the previously mentioned table.

The general structure is the most important, and the gimmick being similar gives a small extra point. An example is Bubble and Heat being similar because the levels are similar, and they both have obstacles in the level that can kill the player in one hit. However, they also have some gimmicks that the other does not have, with the cycling platforms of Heat man and the underwater sections of Bubble man. The numbers used for similarities can be seen in Figure 3.2.

	Air	Bubble	Crash	Flash	Heat	Metal	Quick	Wood
Air	1.0	0.9	0.0	0.4	0.3	0.1	0.4	0.7
Bubble	0.9	1.0	0.0	0.5	0.6	0.4	0.7	0.6
Crash	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.2
Flash	0.4	0.5	0.0	1.0	0.3	0.1	0.5	0.8
Heat	0.3	0.6	0.0	0.3	1.0	0.2	0.4	0.3
Metal	0.1	0.4	0.0	0.1	0.2	1.0	0.0	0.1
Quick	0.4	0.7	0.0	0.5	0.4	0.0	1.0	0.5
Wood	0.7	0.6	0.2	0.8	0.3	0.1	0.5	1.0

Figure 3.2: The table determining the similarities between the different values that the "Level" attribute can have.

The value given from the table will be fed into the global similarity measure as is.

Powerup

The "Powerup" similarity measure is very similar to the "Level" similarity, but has the possibility of having several values at once. This is because the player can have up to 7 of the given suits in the game, as explained in Section 2.1. The values for how similar the different upgrades are can be seen in Figure 3.3.

The similarity values are based on which bosses the weapons are good against, which can be seen in Table 2.2, and how the ammunition work compared to each other. As with the Level attribute, the numbers are based on how the author perceived the weapons to be. The three powerups that also gives access to an item in addition to the weapon gives an extra +0.25 similarity.

Before the value is sent into the global similarity function, the different similarities are summed together and averaged, before they are normalized to get a value between 0 and 1. This is done to ensure that the "Powerup" attribute

	Air	Bubble	Crash	Flash	Heat	Metal	None	Quick	Wood
Air	1.0	0.25	0.1	0.25	0.75	0.1	0.0	0.1	0.0
Bubble	0.25	1.0	0.25	0.0	0.25	0.25	0.0	0.5	0.0
Crash	0.1	0.25	1.0	0.0	0.75	0.75	0.0	0.1	0.0
Flash	0.25	0.0	0.0	1.0	0.25	0.0	0.0	0.0	0.0
Heat	0.75	0.25	0.75	0.25	1.0	0.25	0.0	0.5	0.0
Metal	0.1	0.25	0.75	0.0	0.25	1.0	0.0	0.1	0.0
None	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Quick	0.1	0.5	0.1	0.0	0.5	0.1	0.0	1.0	0.0
Wood	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Figure 3.3: The table determining the similarities between the different values that the "Powerup" attribute can have.

does not dominate the "Level" attribute through having several similarities added together.

3.3 ANN structure and topology

This section describes the topology of the neural networks used as solutions to cases by the CBR system, and how they are trained.

All numbers in this section was found through testing a range of parameters by training them on the Wood man level and see how far it came into the level, as well as how fast it learned to get to that point. Parameters were changed one at a time until the positive trend would halt with the current parameters before moving on to the next. After going through every parameter, the cycle started over again until no improvements could be observed.

3.3.1 Input and output

As input, the ANN takes in a flattened 15x16 matrix, built from what is currently showed on screen for the player, and two extra attributes, representing the health of the player, and the health of the final boss of the level.

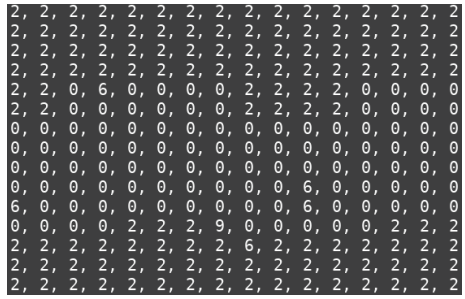
This is then normalized to be between 1 and 0. This was chosen for its reduced size compared to the raw image, and to reduce training time compared to how a Convolutional Neural Network might have solved it.

The conversion from image to input can be seen in Figure 3.4.

The output of the network is a list of all the available commands the player can do:

- Nothing

(a)



(b)

Figure 3.4: The input given to the neural network. (a) shows the raw image, (b) shows what the neural network gets before it is normalized.

- Left
- Right
- Left and Jump
- Right and Jump
- Left and Shoot
- Right and Shoot
- Left, Jump and Shoot
- Right, Jump and Shoot
- Jump
- Shoot
- Jump and Shoot
- Up
- Down

There was originally meant to be outputs for the different suits that the player can chose from as well, but this was removed from the final solution/experiment. This was done because no reliable method was found that would always result in the correct suit being selected without the game crashing. In the end it was decided that the powerup system would get abandoned so that the time could be used on other parts of the system that would be more relevant to answering the main research question, i.e. parameter tuning and training/testing time.

The action that gets the highest predicted reward at each step gets chosen as the action that is performed, as in accordance with the Q-learning approach described in Chapter 2. Other approaches considered were having each button being a single output instead. This would however make it more complicated to generate the target values for updating the weights in the network. Considering this problem, as well as how well Mnih et al. [2015] utilized the former approach, a decision was made to use this approach.

The method used to update the weights was Stochastic Gradient Descent, as this is seen as the standard way to update weights while training neural networks [Mitchell, 1997].

3.3.2 Recurrent nodes and hidden layers

Between the input and the output layer are three hidden layers, and five recurrent memory cells, so that the network remembers the last five states it has experienced.

These nodes are placed between the input layer and the first hidden layer. Five recurrent cells was concluded to be the ideal number, as any more only increased the time needed to train on each iteration, but did not have any impact on the performance of the agents. Having less than five however, did affect the performance of the agents negatively.

The three hidden layers has 1000 nodes each. Having less than three hidden layers made the network not able to properly learn how to behave, while having more than three didn't have much of an impact. The same can be said for having less than 1000 nodes in each layer. Having more than 1000 did, however, show hints of improving performance if more time was available for training.

Larger networks in terms of nodes in the three hidden layers did learn more advanced behaviour, but at the same time required more time for training than the smaller networks before getting to the same level of performance.

It was therefore decided to go with the smaller 1000 node hidden layers, as the time needed to train the networks had to take priority.

3.3.3 Error function and Activation function

The activation function is the same throughout the entire network, except the output layer. The four activation functions mentioned in Chapter 2 were considered for the model: sigmoid, tanh, ReLU and ELU. They all got to the same level of performance on the problem eventually, but ReLU and ELU came to the same level faster than what sigmoid and tanh could. The choice therefore came between ReLU and ELU, in which ELU was chosen, as it was slightly more consistent over several runs compared to that of ReLU. This gives the activation function:

$$f(x) = \begin{cases} x, & \text{if } x < 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

And its derivative for the update rule:

$$f'(x) = \begin{cases} 1, & \text{if } x < 0 \\ f(x) + \alpha, & \text{if } x \leq 0 \end{cases}$$

Where x is the weighted sum of the previous layer, α is a constant value $0 < \alpha$, which in this model was set to 1. The final output layer used a linear activation function, so that it would be able to predict any value for the reward.

The error function used in this model was Mean Squared Error:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - T_i)^2$$

Where n is the number of output nodes, Y is the predicted value and T is the target value. Squared error was also considered, which does not divide by n, but is otherwise the same. It did however have issues with the overestimation problem mentioned in Chapter 2, and was therefore not chosen over Mean Squared Error.

3.3.4 Q-learning update

This subsection describes how Q-learning is handled for the model, how targets are generated, how the reward function calculates the reward given at each step and what is done to combat the overestimation problem of Deep Q-learning.

Generating target values

The target values are generated based on the reward given for a certain action and the maximum estimated value. This value is then put into the original prediction, ensuring that the only difference is the action that was tried for that instance.

All the actions that were not taken gets the same value it had previously. If the state was terminal, i.e. the player died, it will only get the reward as the correct value, and not the reward and the maximum estimate.

The algorithm can be seen in Algorithm 1.

Algorithm 1 The algorithm for generating the target values

function GENERATETARGETVALUE
$a_s \leftarrow action\ in\ state\ s$
$targets \leftarrow Q(s_i, a_j) for\ all\ a$
$target \leftarrow \gamma max(Q(s_{i+1}, a_j) for\ all\ a) + r$
$targets_{a_s} \leftarrow target$
$return\ targets$

Reward function

As the goal of all the levels (see Section 2.1) are somewhere to the right of the starting point, the main bulk of the reward comes from getting movement to the right. When the player moves right or into a new screen, the value representing the position on the x-axis in the levels increase.

The idea for giving extra points by going into a new screen is to encourage the agent to find the new screen, even if it means losing some points in the short term.

In addition, it gets heavily punished for deaths, so that it should never be worth it to choose death over staying alive.

It also gets punished for not having any movement, more so than if it were to move left instead of right. This is for the agent to not get stuck in a local minima and never move left again.

Lastly it gets reward for taking health away from the boss of the level, and it gets heavily rewarded for killing the boss, which completes the level.

The full function can be seen in Algorithm 2.

The value returned is divided by 100, to keep the overall values from growing too large, as this sometimes resulted in overflow errors. It is not normalized completely because there is no known maximum or minimum amount of points in a given level.

As with the hidden layers and recurrent nodes, the reward function was tested by changing, adding or removing a single reward at a time, and see how the behaviour evolved for the Wood man level. This was repeated until no improvement could be identified. Having the reward for moving to the right too large means it ignores everything and just wants to hold right forever. Too big punishment for deaths made the agent afraid of moving from the starting line, too small made

it not care if it died or not. No punishment for standing still and moving left made it more likely to find local minima where it could just stand in a corner somewhere without dying.

Algorithm 2 Algorithm for generating the reward at each step

```
function GetReward(prev_x, prev_screen, prev_boss, best_x)
    boss_life ← Current boss life
    x ← Current x position
    screen ← Current screen number
    terminal ← Is state terminal
    if terminal then
        reward ← −510
    else
        if x = prev_x then
            reward ← −10
        else
            reward ← x − prev_x + max(x − best_x, 0) + max(screen − prev_screen, 0) ∗ 510
    reward ← reward + max(prev_boss − boss_life, 0)
    if prev_boss > 0 and boss_life = 0 and terminal = false then
        terminal ← true
        reward ← reward + 1020
    if x > best_x then
        best_x ← x −
         return reward, x, best_x, screen, terminal, boss_life, life
```

Double DQN

Overestimation is a big problem with Deep Q-learning, as mentioned in Chapter 2. The method that was decided to combat this problem in this model was Double DQN [Van Hasselt et al., 2016].

Having a target network that is not updated at every training step vastly improved performance compared to a more basic DQN, which often eventually just held a single button combination forever, no matter the situation.

Meanwhile the agents trained with Double DQN could still have varying behaviours in the same levels, even when trained for longer periods of time. One other method was considered for combating overestimation was a Dueling Network architecture. This method should outperform the Double DQN, but was not implemented because of time constraints. This will be discussed as a possible improvement in Section 5.4.

Chapter 4

Experiments and Results

This chapter contains all information for the experiments used to test the CBR-ANN hybrid system outlined in the previous chapter.

It starts with the experiment plan, which describes how the experiment will be performed, and how the CBR-ANN hybrid and baseline ANN agents are evaluated. Following the experimental plan is the setup, containing all the parameters needed to recreate the results. Finally the results will be presented.

4.1 Experimental Plan

This subsection contains all information about the experiment used to answer the research question presented. It starts by presenting information on how the training and testing will be done, and how the models will be evaluated. To finish it off there is a section with predictions of how the CBR-ANN hybrid system will compare to the baseline ANN.

4.1.1 Training and testing

This subsection will take on which cases are used for training and testing and why, as well as explain how the system will be evaluated. All experimenting will be done on the American release of Mega man 2, and it will be played on Normal difficulty. The game runs at 60 frames per second.

Training

For training, there are five cases used, being five different levels in the game presented in Chapter 2. Each case will get the same amount of training steps.

The networks used by the CBR-ANN hybrid and the baseline ANN will all have the same topology.

The CBR-ANN hybrid and the baseline ANN agent will train on the five cases in the same order and same parameters. The first network in the CBR-ANN system and the baseline ANN will both start with random weights, as this is the standard initialization method used for weights used in ANN [Mitchell, 1997].

Testing

The testing will take place on all of the eight mentioned levels in the game, from the level select screen. The CBR-ANN hybrid and the baseline will get a reduced amount of steps compared to their training-steps and play out these steps on each level in random order.

The two agents will then be compared on how they do on the following criteria:

- Total reward gathered for each level and in total.
- Best reward in each level.
- Number of deaths in each level and in total.
- How their behaviour compare to how a human might have played the levels.

This direct comparison should do well in answering the main goal and research question of the thesis. This experiment will directly answer the research question, which in turn helps in exploring if the proposed method is suitable for these kinds of environments.

4.1.2 Expectations

Expectations for the results are that the CBR-ANN system will do better on average, especially on the more unique levels, and the levels that was trained on early on, as the baseline is more prone to forgetting about what it previously learned when it has trained on other levels more recently.

As for the unseen cases it is harder to predict, because it's hard to say if the similarity functions will be good enough that it chooses the appropriate network, and that this network fits the problem, or if it's overfitted for its previous example.

The baseline might turn out to perform better for unseen cases, as it has been trained for multiple scenarios, and might therefore be less prone to overfitting. The training method used here however might hinder it in being overfitted for the last level it sees, as it plays out all if its training steps for one level in one go rather than mixing up the levels more, and seeing the different levels several times.

4.2 Experimental Setup

This section presents all the parameters and necessary data relevant to repeat the experiment presented above.

It starts with the generic parameters, meaning parameters that doesn't necessarily do anything for the neural network, or the game specifically, then there are the parameters for the game, and last is the parameters for the neural networks. The parameters used are the same for both the CBR-ANN hybrid and the baseline agent.

4.2.1 Generic parameters

The generic parameters are how many training and test steps will be performed, how many times each level is repeated, how big the memory is and the order of the training and test cases:

Training steps per level per repeat 100000

Test steps per level per repeat 10000

Repeat level 1

Number of situation remembered 1000

Training order The training was done in the following level order:

1. Wood man
2. Flash man
3. Quick man
4. Metal man
5. Air man

Test order Random

All the parameters were found by doing tests with different values in the same way as described for the network topologies and rewards in Chapter 3, except the order of the cases and number of times the level is repeated. 100000 training steps was the minimum amount of steps needed to get consistent results on the different levels over several runs.

Test steps are heavily reduced compared to training steps because it doesn't need as many steps to show how well it does in the levels compared to how much is needed to train the agents.

The model should get better with more training iterations, but it was decided against giving it more steps because of time constraints.

Repeating the level is also set to 1 for time constraints. Putting this to above 1 should have little to no effect on the CBR-ANN hybrid compared to increasing the training steps, but could potentially help the baseline agent with overfitting.

The number of situations remembered for generating data to train on was decided to be 1000 because it gave the most consistent performance at low amounts of batch numbers for the neural network, which was needed to get training done within the deadline. The batch numbers and other neural network parameters can be seen in Section 4.2.3.

The order was decided by having the most generic level, that the agent became the strongest at most consistently. The following levels are in random order for the CBR-ANN hybrid, and then the baseline agent mimics the order given to the CBR-ANN hybrid.

It started on the most generic and easiest level to have a higher chance of developing a good foundation before going into the other levels.

The order of the testing levels should not matter for the performance inside each individual level, and is therefore set to random.

4.2.2 Game specific parameters

Game specific parameters are parameters that affect how the agent interact with the game environment. They are how many times an output from the neural network (action) is repeated. The chance of a random action being taken during training instead of the estimated best action.

The random action will also be set to 1 if the best x position is not improved over a certain number of steps, or if there are no movement at all over a slightly smaller time frame.

If there are no improvements after another threshold, it will be treated as a death.

Actions repeated 5 frames

Random action chance 1 before learning starts or if no improvement threshold reached, otherwise 0.7 as the base value

Minimum Random action chance 0.1

No movement threshold 200

No improvement threshold 500

Inactive threshold 1000

Lowering the repeated actions makes the problem more complex, with more complex movement available to the agent, but it also makes it harder to learn

the more simple things like jumping. How high the player can jump depends on how long the jump button is being held down after the initial press of the button. If the action is to be determined on every frame, the chance of another action interrupting the jump is higher, making it less likely to learn how to jump over e.g. ledges early on.

Given enough time, having a lower number of repeated actions might be better, but for the time available, 5 was the most consistent and still being able to get slightly more complex behaviour than any higher values would give.

The random actions starts as 100% random, and stays high for a while to have the agent explore more in the beginning when it knows very little about its environment. The random settings are reset to the default every time the agents starts a new level.

For each time step, after learning has started, the random actions are reduced by $base\ value * \frac{current\_step}{total\_steps}$, so that it can learn more about what it thinks is more optimal in the later stages of the training. This is to encourage exploring early in the learning process, while exploiting what it learned and improve on this later in the process. The chance of a random action can never go below the minimum random action chance parameter.

The three thresholds are to combat local minima that the agent often finds, by forcing it to explore more if it gets stuck, and eventually treat being stuck as a death. Having them too high makes them very ineffective, while having them too low makes the agents unable to exploit their own inference.

4.2.3 Neural network parameters

These parameters are the ones that affect the neural network and its updates. They include the learning rate, discount, when it starts to learn, batch size, number of batches, target q-network updates, update frequency.

Learning rate 0.1

Minimum learning rate 0.01

Discount 1

Step start learning 1000

Batch size 128

Number of batches 2

Update target q-network 10000

Update frequency 30

The learning rate starts high because it did learn properly in the beginning with lower rates, with the available learning steps. Like with the random actions, the learning rate decreases over time with $base\ value * \frac{current\_step}{total\_steps}$ and cannot go lower than 0.01.

The discount is set to 1, meaning that it will always consider all of the rewards it will get in future steps during the Q-learning update. This was the most consistent and didn't get stuck in local minima as much as the more greedy agents that was experimented with.

The network starts updating after 1000 steps of random actions to combat overfitting the early rewards. After every 30 actions done, the agent trains on the two batches of 128 previous states from its memory.

In each batch, half are the most recent states, while the rest are selected at random. The number of batches are low, and the system should greatly benefit from having many more than the ones given currently, but generating the target values and batches took a lot of processing time, so it was reduced keep within the time available.

Having just 2 batches still gave consistent results, but having more might have made the agents learn more complex behaviour, and a greater chance at not forgetting an action in certain states that might be a rare occurrence, which did become a problem.

Every 10000 step, the target q-network is set to become the same as the network doing the forward pass to choose an action. Lowering this gave problems with overestimation.

4.3 Experimental Results

This section contains all the data collected during the testing phase of the experiment. It starts by describing how the CBR-ANN hybrid and the baseline agent interact with the environment in general, then goes on to present the rewards collected, and compares how much they die. Finally it compares how the two agents behave with how humans might play the game.

4.3.1 General behaviour

The general behaviour for the CBR-ANN hybrid is to get to the possible position it can find early in the level, and make sure to die before it's possible for it to get stuck. This was either done through taking consistent damage over time (Wood man) or jumping into a nearby pit (Air man).

This behaviour get penalized heavily compared to getting further in the levels. The agent did however have trouble finding these alternatives while exploring and instead got stuck in this pattern. The overall negative value of waiting

until timing out is much larger than the negative value of a fast death, so it is understandable that it would rather die fast than time out. This behaviour gave better overall rewards in most of the cases seen.

The baseline is afraid of the pits, and runs left to avoid death in almost any level, and dies from not improving its position in the given time frame. Instead of the longterm gain that the CBR-ANN hybrid learned from having a fast death, the baseline ANN preferred to wait to get the short term reward of not dying. It did however learn to go right and left some instances where it was a suitable sequence of actions to take, which the CBR-ANN hybrid failed to learn.

4.3.2 Rewards

The reward collected throughout the entire testing and training stage can be seen in Figure 4.1. The curve for their training follows a very similar pattern for most of the duration, but the total reward collected by the CBR-ANN hybrid during training is much larger than the one of the baseline ANN agent.

The similar pattern end after the fourth training case is done, and the baseline agent can't seem to recover from what looks like overfitting or overestimation for the two previous levels, which both gave a reward for going the opposite direction of what was required in the last level.

For the testing, this case of overfitting or overestimation was fatal in its comparison to the CBR-ANN hybrid.

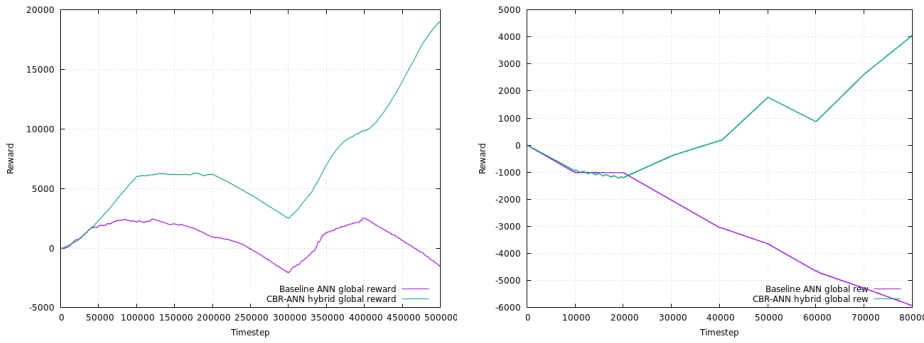
The two agents did not do the levels in the same order for the testing phase, as the order was chosen at random, so the fact that they are similar in the start does not mean that they scored this amount for the same case.

The interesting part of the testing results are the massive difference by the end. The CBR-ANN is in the positive as far as the reward was concerned, while the baseline ANN's problem with overcoming the problems it encountered with overfitting/overestimation made it get an almost linear negative reward.

Bubble man was not in the training set, and is interesting to look at for the performance of the CBR-ANN hybrid, considering how it was predicted that it might have trouble with unseen cases.

This is because of how its cases are more tailored towards one behaviour for each level. It did however not have any trouble collecting a bigger reward than the baseline agent in this level, as can be observed in Figure 4.2.

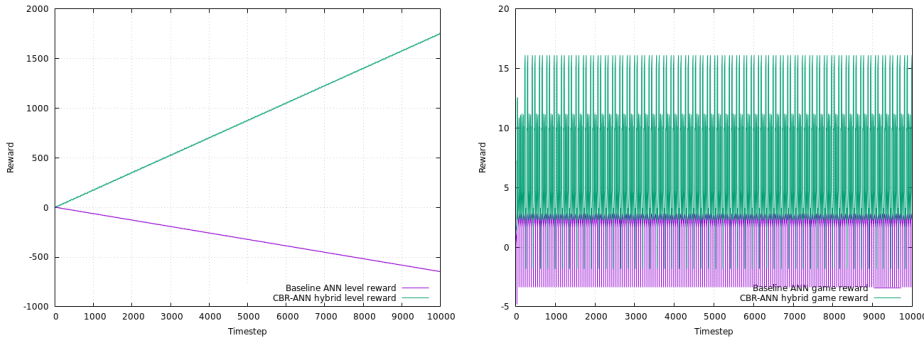
The behaviour of the agents here consist of the CBR-ANN running right for as long as it can, and then run off a cliff to its death, instead of trying to jump to the next platform, while the baseline runs left and dies at the start, showing signs of its overfitting/overestimation problems. The pattern seen in Figure 4.2(b) shows this rapid death pattern in both agents.



(a) (b)

Figure 4.1: The reward collected through both the training and testing stages. (a) shows the reward collected during training while (b) shows the reward collected during testing.

The CBR-ANN hybrid still collects a positive score for this level, even with its meany deaths, which will get discussed in the following subsection.



(a) (b)

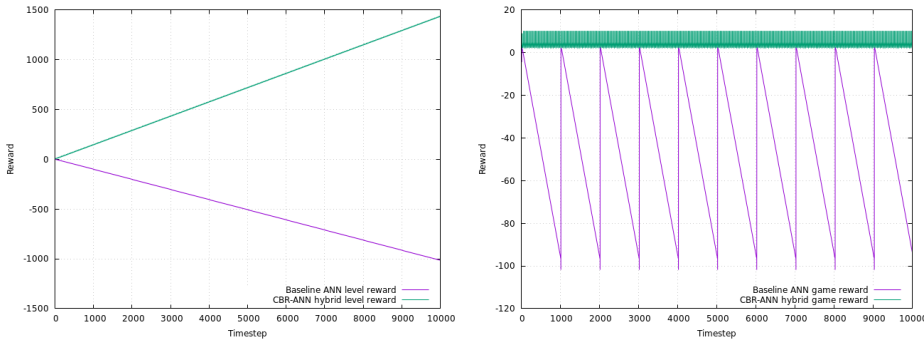
Figure 4.2: The reward collect in the Bubble man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

The next level compared is Air man, which was in the training set. This level was the most similar to that of Bubble man in the CBR part of the CBR-ANN hybrid.

Again, the CBR-ANN hybrid's performance is vastly superior to that of the

baseline ANN. The CBR-ANN hybrid doesn't perform as well as it did in Bubble man, as the cliff it dies from jumping into was earlier in the level.

Meanwhile, the baseline goes left in this level as well, and gets stuck against a wall, causing it to get a lot of negative points for not having any movement. The rewards can be seen in Figure 4.3.



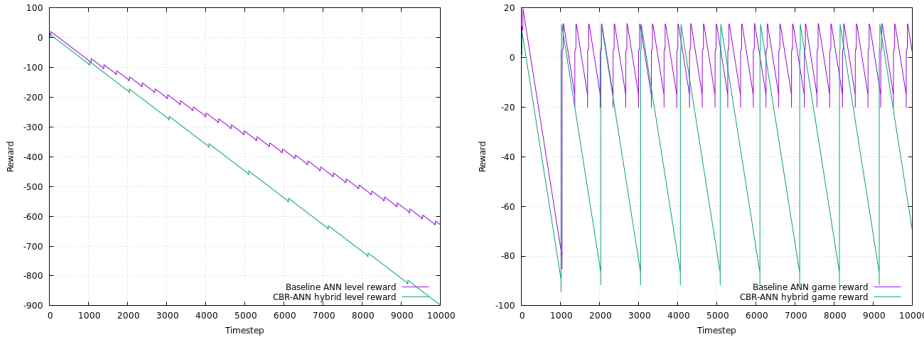
(a) (b)

Figure 4.3: The reward collect in the Air man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

Quick man is one of the more distinguishable levels, having the player go both right and left, getting most of the reward by falling down. Both of the agents perform rather poorly in this level considering the reward they gathered, which can be seen in Figure 4.4. Both the agents get stuck in this level, however the baseline gets stuck further into the level, causing it to get a slightly higher score than the CBR-ANN hybrid. This is one of two levels where the baseline performs better than the CBR-ANN hybrid. As both agents trained on this level without, and the CBR-ANN hybrid did not perform to the same level or better, this might be a case of the random actions. The baseline ANN might have been "lucky" and discovered that falling down the pit on the left gave a higher reward, while the CBR-ANN hybrid never happened to see this pattern, or it didn't see it enough times to learn it.

Wood man is maybe the most generic and easy of the levels to play, as there are no death pits and it's mostly running right. The CBR-ANN hybrid performs quite well on the first section of this level, but during training got stuck when trying to get to the section after, because it had to turn left to get there. This then caused it to instead take damage throughout the entire duration of the first section, and die as it hits the end.

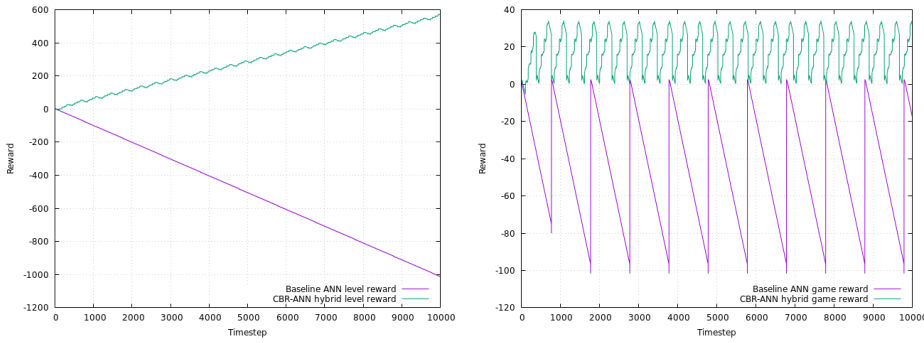
The baseline agent has the same performance as it got in most of the levels, with just running left and get stuck. The reward can be seen in Figure 4.5.



(a) (b)

Figure 4.4: The reward collect in the Quick man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

As can be observed in the reward per game played, it is relatively high compared to the other levels, as the agent gets very far to the right before dying. This is the level where the agent looks most like what a human player would play like.



(a) (b)

Figure 4.5: The reward collect in the Wood man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

Crash man is another highly distinguishable level compared to the rest, and it was not part of the training levels. It is, not surprisingly, one of the worst levels for the CBR-ANN hybrid. This is not surprising because the CBR-ANN hybrid consist of a collection of experts on the cases it has seen, and this level

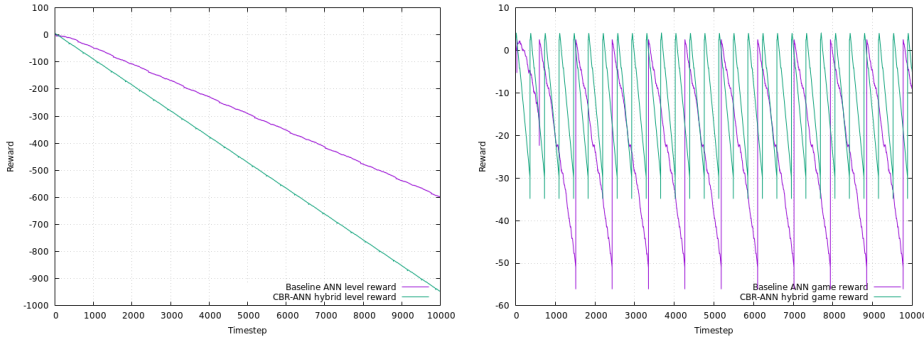
looks nothing like anything it has previously experienced.

The baseline ANN also performs badly in this level, but it does better than the CBR-ANN hybrid. The rewards can be seen in 4.6. The level is mostly climbing up, which none of the agents has been able to learn in the training stages, as no other stage requires this.

This is also something that the agents are highly unlikely to learn because of how the reward function works with giving a big punishment for not moving horizontally.

They both get stuck on opposite ends of the starting area, but the baseline ANN gets stuck further away from where enemies spawn, making it survive for longer and therefore not get punished as often.

It can be seen by the peaks of the game rewards for the two agents that the CBR-ANN hybrid gets stuck in a better position, but because it dies more often, it gets a total reward that is lower than the total of the baseline ANN. This seems to be because of the single ANN's preferred strategy in all levels is to move left, while the CBR-ANN hybrid prefers going right, which in this one case happens to be the worst case scenario for the CBR-ANN.



(a) (b)

Figure 4.6: The reward collect in the Crash man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

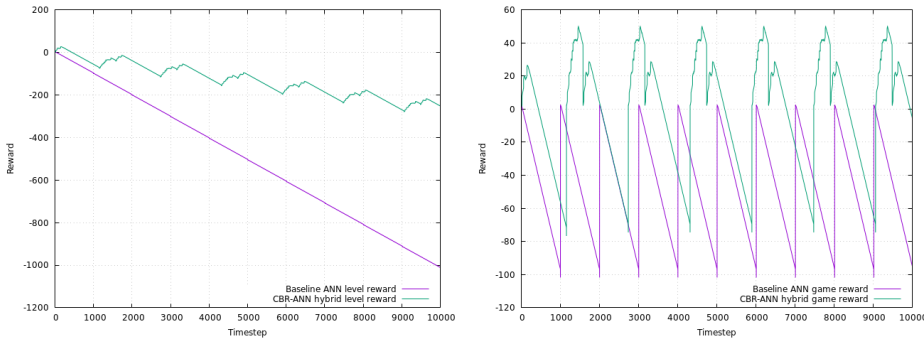
Flash man's layout is rather generic, but has some hard parts for the agents to learn with its more maze-like structure, having many local maxima positions where the agents can potentially get stuck.

In addition, the movement is slightly different with having the player slide around with its momentum, which does not happen in the other levels. The rewards can be seen in Figure 4.7.

The peaks of the CBR-ANN is higher than in any other level, and it has two different patterns it follows in this level.

One of the patterns it gets stuck and gets a massive total punishment, while the other pattern it finds an enemy that can kill it as it finds a local maxima, leading it to not get as big of a punishment as when getting stuck. The total however is still negative.

The baseline ANN has the same pattern as it has for most levels with just going left and getting stuck in the beginning until it dies.



(a) (b)

Figure 4.7: The reward collect in the Flash man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

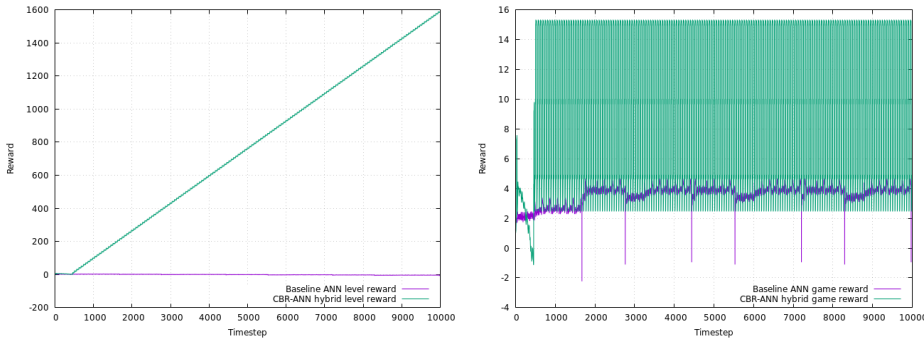
Metal man is another highly distinguishable level. The structure is similar to that of Air man and Bubble man, but it has conveyor belts, making the player move left or right by itself when no inputs are given, and accelerates or hinders movement while trying to run in a direction.

This level saw a more advanced behaviour pattern in both agents, having them try to avoid the first pit for a short amount, though not jumping over it. Their behaviour is similar in this level, though the CBR-ANN hybrid outperforms the baseline ANN. The rewards can be seen in Figure 4.8.

The last level, Heat man, was not in the training set. Its structure is similar to that of Air man, Bubble man and Metal man, but has a gimmick where there are platforms that fades in and out of existence at set time intervals, however none of the agents made it this far into the level.

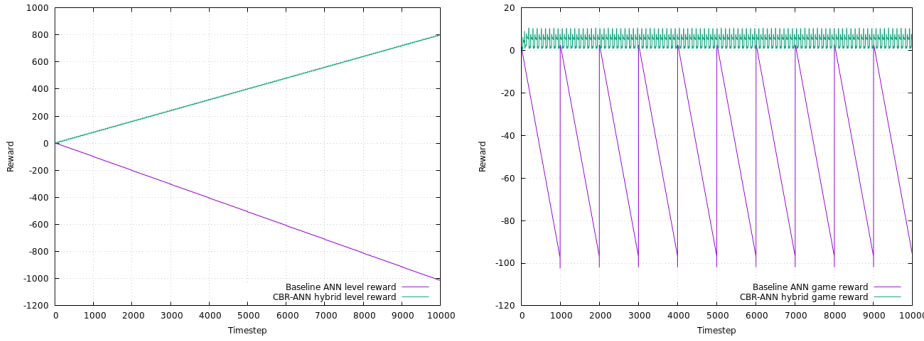
As usual, the CBR-ANN hybrid goes as far right as it can, before it dies in a pit, while the baseline ANN gets stuck on a wall to the left of the starting point. The rewards can be seen in Figure 4.9.

The pit in this case is very close to the beginning, making the CBR-ANN hybrid gather a pretty small reward per game, but the total is still positive.



(a) (b)

Figure 4.8: The reward collect in the Metal man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.



(a) (b)

Figure 4.9: The reward collect in the Heat man level. (a) shows the total reward collected while (b) shows the reward in each game played, resetting every death.

4.3.3 Deaths

The deaths the two agents accumulated throughout the testing process can be seen in Table 4.1. The CBR-ANN hybrid accumulated several times more deaths during the testing, even though it performed better in almost every level.

This shows that even though the baseline survived for longer, it doesn't really matter for its performance as much, as getting stuck for a long time and dying of time out is a lot worse for the agents than to get as far to the right as it can before dying without getting stuck, like the CBR-ANN hybrid learned to do for

almost all the levels.

Table 4.1: Table of the deaths the two agents collected throughout the testing phase.

Agent	**Bubble man**	**Air man**	**Quick man**	**Wood man**	**Crash man**	**Flash man**	**Metal man**	**Heat man**	**Total**
CBR-ANN hybrid	271	294	9	25	27	12	163	93	894
baseline ANN	192	9	28	10	12	9	7	9	276

4.3.4 Compared to human behaviour

The agent's behaviour would look strange to how a human would play in most of the levels. First off, a human would normally try to kill the enemies it sees to reduce the chance of dying, while the agents ignores them and the consequences of taking too much damage entirely, and just tries to get as far as it can before it dies.

This is most visible in the Wood man stage. Where a human might take it slow, kill every enemy in its path and get to the end of the first section with a lot of health still remaining, the agent takes damage at every enemy it encounters, that blocks its path just to get to the end as fast as possible.

The end also requires another input than to run right, which might also explain why it doesn't try to avoid damage, as it sees it as better to die just as it reaches the edge, rather than getting stuck there.

As for the other levels, a human would try to jump over the pits and avoid dying, while the agent runs or jumps off the edges without really trying to reach the other side.

A human would also be able to see the walls and react accordingly while both the CBR-ANN hybrid and the baseline agent would run into walls forever if they think there is more points to get by going that direction, in the CBR-ANN's case, it was right, while the baseline agent seemed to have overfitted on the few levels that requires going left.

Overall, the behaviour compared to how a human would do the levels, the agents looks very underdeveloped. This and possible solutions to the problem will be discussed further in the following chapter.

Chapter 5

Evaluation and Conclusion

This section will discuss and evaluate the results from the experiment, its contributions and come to a conclusion for the goal and research question based on the results. It will also discuss some limitations that came up throughout the project, and finally come with suggestions for future work, both on how to possibly improve the current setup and some bigger changes that could be possible.

5.1 Evaluation

As for the expectations presented in Chapter 4, most of them were correct when looking at the results, but not all of it went as expected.

As expected, the CBR-ANN hybrid did better overall than the baseline ANN, but not necessarily on the more distinguishable levels. Some of the more distinguishable levels actually got better performance from the baseline, even on the cases that were in the cases that were used during training.

The biggest surprise here was the Quick man level, where the expected outcome before starting the training and tests was that the CBR-ANN hybrid would learn better that it had to alternate between going right and left.

However, the CBR-ANN hybrid only ever went right in this level, while the baseline ANN did right at first and then changed to left, getting a better total reward than the CBR-ANN hybrid.

The other special case where the baseline outperformed the CBR-ANN hybrid was the Crash man level, this level however shouldn't mean too much overall, as even if the baseline did better, they both got stuck in the first section of the level until they either timed out or died from damage taken from enemies. The CBR-ANN happened to be stuck closer to where the enemies spawned, so it died

more often. This level was also not part of the training set, meaning none of the agents had ever seen something similar to this level.

The expectation that the CBR-ANN hybrid would be more susceptible to overfitting proved to be a false prediction. The CBR-ANN hybrid performed well on both the training set and the test set compared to the baseline ANN in almost every case, and adapted well to the new cases it had not seen before. Meanwhile the baseline ended up only doing one set of actions on almost every level, leading to the same reward pattern in almost every single one, which can be seen in the reward figures. This is also very visible in Figure 4.1, where it clearly starts overfitting in the fourth level, and never recovers during the fifth case.

In a vacuum, the performance of the CBR-ANN hybrid is not particularly strong. It beats the baseline in almost every case, but it does not get very far in any of the levels that were part of the experiment, including the training cases.

Its best level in terms of how far it got was the Wood man level. It got a higher reward in Flash man, but it was further away from the end of the first section here.

Neither of these are any far into the levels either, but it's better than dying at the beginning of the stage. It also never learned that dying was not a good thing, as it would rather die quickly than get stuck somewhere before dying.

This made the CBR-ANN hybrid not always advance in the level, as it learned that it should not get into certain situations, rather than getting into that situation again and do another action than it tried last time. This is not visible in the testing results, but could be observed while the agent was training.

The number of deaths are rather disappointing, as it should get a rather big punishment for dying, but it still thinks dying early is better than getting stuck later on.

To evaluate whether the goal of exploring if a hybrid CBR-ANN solution is a viable option for working in a dynamic environment with large hypothesis spaces has been accomplished or not, the main research question needs to be answered. The CBR-ANN hybrid performed well compared to the single ANN in the selected dynamic, changing environment.

In reward collected, the CBR-ANN hybrid outperforms the single ANN in six out of eight levels. The CBR-ANN hybrid also learns more advanced behaviour for several levels compared to the single ANN, that only uses a single behaviour for all levels outside of one. The single ANN forgot the behaviours it used in the earlier examples experienced during training, while the CBR-ANN could recall the level specific behaviours it had previously experienced. This is more in line for how a human would learn to play the game, as a human wouldn't necessarily forget how to interact with a previous case just because the behaviour used in a more recent case was different.

Deaths however did not seem like an issue the CBR-ANN learned to deal with, as it would rather get as many points as possible and die instead of getting stuck. The single ANN however preferred to stay alive until it timed out. The CBR-ANN strategy of not being afraid of deaths did make it perform better in accordance with the evaluation criteria, it should still not be a desired behaviour.

Outside of the evaluation criteria presented, neither agent did perform particularly well. In most of the cases, neither agent got past the first section of a level. In most cases, the CBR-ANN hybrid did get further in the first section of the level, or was the only of the agents to get past it.

The results gathered for the research question suggests that the CBR-ANN hybrid is suitable for dynamic environments with large hypothesis spaces. More research is however needed before this can be answered with confidence. To get closer to the goal, more comparisons should be made, both with the suggested single ANN, as well as potential other methods. As neither agent performed particularly well in vacuum, it would be interesting to see how other methods would solve the problems compared to the CBR-ANN.

Suggestions to improve the evidence of the results as well as weaknesses with the suggested methods and potential changes that could improve the CBR-ANN hybrid are presented and discussed in the following sections.

5.2 Discussion

The strengths of the CBR-ANN model is in its CBR part and how it chooses a network that should fit the situation, rather than try to fit one solution to all situations. It does well relative to the baseline it is tested against.

The baseline, however, did not train under optimal conditions for itself, as it most likely would do better if the total number of steps were the same. Instead of doing one level until all training was done for that instance, and then move on to the next, it could potentially benefit from mixing the levels or cases more in smaller intervals to counteract overfitting. For the CBR-ANN this would make little to no difference, as it would just load up the network it previously used for that case and alter the solution for this case again to make it fit better.

A weakness of the model is how it might learn a behaviour in one case that could be useful in several cases, but is not transferred since the solutions in the case base does not communicate in any way. This might be the biggest problem for this method if it's used for bigger problems and case-bases than what were used in the experiment in this thesis. This potential problem was not very noticeable in this experiment, but could become an issue when the case-base grows larger.

One of the reasons this experiment can not answer the goal of the thesis with confidence is that it only had one learning session, making it possible that the agents was just lucky to get these exact results. This can be seen in e.g. the first

training example in Figure 4.1, where already in the first level, the CBR-ANN hybrid outperforms the ANN, even though there should not be a difference this early on. The results would be much stronger if it was using the average of several agents' results rather than just the result of a single training and testing period.

One of the main reasons that the results are not strong enough to answer the goal of the thesis confidently is because of time problems as the project went on. The agents, both the CBR-ANN hybrid and the baseline uses a lot of time to learn certain behaviours, and might not have gotten enough time to explore the levels properly, because of the time constraints mentioned several times throughout the thesis.

Time became an issue because of a bug found very late in project that severely hindered the agents' ability to learn anything from its experiences. It was discovered very late because of how it looked like the agent was improving over time, when in reality it was the chance of random actions going down, but still being there, coupled with a lucky initialization of weights, making the agent naturally go towards the right.

This made it look like it improved as the random action chance was going down. A lot of time was therefore wasted trying to find optimal parameters for this broken system, and when the error was found, there was not enough time left to properly test new sets of parameters properly.

This makes it possible to potentially vastly improve the performance of both the CBR-ANN hybrid and the baseline agent to perform a lot better with more time for testing parameters and training the final agents. This time problem was also the cause for the test only being performed once once, rather than e.g. training several agents and compare the average.

5.3 Contributions

This thesis contributes with research on a method that has not been previously explored in the way presented. It performs better than the methods it is compared against in almost every case, both in this thesis and in similar methods discussed in Section 2.4.

It provides a reason to further explore this method, as its performance outperforms the method it was compared against, both on cases they both trained on, and cases that was not explored beforehand. There is however still a lot of work to do to get satisfying results for this specific problem and to expand further on the goal set for the thesis.

The result itself shows potential compared to other methods, and with a good possibility that this could be vastly improved with more time for finding parameters and doing tests. This model might prove to be of greater significance in the future when more research has been conducted.

This thesis contributes with a strong basis for further research on the method suggested, presented in Section 5.4.

5.4 Future Work

This section discusses some possible changes that can be done for future work on this model.

It contains both possible solutions for the problems described over with lack of testing, extending the case base and the current method, as well as some completely new methods that could improve the performance.

5.4.1 Extend current method

The obvious first step would be to extend the current method, so that it can solve the problem in a more satisfying way, and reach a better conclusion to the goal and main research question.

First would be to find better parameters for the neural networks. As mentioned, the bugs found very late in the project reset all progress on the parameters found. Some testing could still be done, and some trends was found, which was discussed in Chapter 3 and 4. However, bigger neural networks, different placements of the recurrent nodes and especially, longer periods of training time for it to learn the levels better should all be explored more, with more time available, as they could improve the performance drastically.

Other ways to change the current model would be to change the inputs. Both giving the network more information than it already gets, and changing the values of the different objects that the agent can see. Currently the values are very similar to each other, it might have been better for the network if there was a bigger difference between them, to easier be able to distinguish the enemies from the background and the player.

The reward function could also benefit from some work. Some possible changes could be to find other ways of punishing getting stuck, have punishments for health going down or bigger punishments for dying.

These were briefly tested, but more advanced ways of punishing getting stuck made it hard for the agent to learn what made it lose the points. Having too large of a punishment for dying often made the agent afraid of going near places where it could die. Punishment for losing life gave the same result.

No points for not moving and for moving left was also tested, but often resulted in the agent standing still at the start eventually, not having to worry about dying there. Even though they were tested, more testing could still be done, and with different ratios. It might have been that it didn't get enough time to adapt to the bigger death and health penalties.

Other things that could be tested could be points for killing enemies or collecting extra health. Making the reward for going left/right reversed if the agent stands still for too long could also be a possibility.

Changing it from a basic recurrent neural network like it is now and into an LSTM could also be a possibility, as well as changing how the network remembers. Currently the network gets as input the number states it keeps in memory at every timestep.

This was done to keep it more consistent with doing random samples of training data while testing, so that this would not overlap with the memory it builds up while playing. It handles it by first going through the first input and uses this in the next input and so on until it reaches the last one. It did not make a big difference in the testing done, but might be worth trying again.

Another change that could have a big impact would be to increase the number of cases that is used to train the model at each training step. Instead of only having two batches, it could get hundreds or thousands per step.

This could prove to give a massive boost in performance, as it wouldn't as easily forget the older cases that might have been positive, but that it doesn't see that often because the model hasn't gotten to a level where it can consistently reach that situation again. However, this would also increase the training time needed significantly, as making the batches and target values is a rather slow process. Given more time, this should be one of the top priorities for improving the current model.

All of the mentioned suggestions, except the last few on the reward function, had been tried before the setback from the bug mentioned earlier, but unfortunately there was not enough time to experiment properly after the bug was identified and repaired.

5.4.2 Extend current problem

Another obvious task for future work would be to extend the problem again. As mentioned, it was originally meant to include the powerups as well as the levels, but had to be cut down to only levels because of time constraints.

The reason for this was both the bug mentioned earlier and another problem with the game crashing when using the most reliable method of switching between suits, as it did sometimes end up in situations that were not allowed by the game.

The way it was implemented was writing into memory which powerup it was supposed to use, and change it that way. This made it possible for it to change which suit it used while the effect of other suits was still active, making the game crash as this was not behaviour that would normally be possible.

To make it possible to implement this it would either need some kind of logic to make it impossible for the suits to overlap with writing to memory, some logic

that can generate the necessary inputs to choose the correct powerup consistently, or to change the input and output to allow the agent to see the menus as well as giving it extra possible outputs to be able to get into the menu.

All of these solutions were possible time sinks, and therefore it was elected not to implement them, as the goal and research question could still be answered with the smaller set of cases. It should however be a priority to implement this in future work, as it might be able to show the differences even better than when there are as few cases as there were in the example here.

Other ways to change the problem could be to divide the levels into several cases, e.g. one case per screen in the game. This could potentially solve at least Wood man's stage without changing too much of the rest, but would require a more advanced similarity function, as comparing sections of levels to each other might be hard compared to comparing the levels as a whole to each other.

Having a table lookup method like the previous one might work, but would get very cluttered and hard to work with, as there is a large amount of screens to consider in each level, so another method would be preferable.

Lastly, it could get to use the life system of the game or play the game from start to finish, choosing which stage to do next by itself. This could however extend the training time significantly, and potentially not be able to train on several different levels, as it could get stuck on the same level throughout the entire process. This would make it not able to really show the difference between the CBR-ANN hybrid and the baseline.

This could be considered after the cases above have been successfully experimented with, and it is knowns that it can reach the end of the levels eventually.

5.4.3 Various ideas for extending the model

This subsection will have a collection of possible future work that is more loosely connected to the model, and in some cases might alter it drastically. Some of the changes suggested here for the CBR part might need a bigger case base than only the levels to properly show their results compared to the current method.

A rather simple suggestion would be to instead of having a random initialization, to start the network for both the CBR-ANN hybrid and the baseline with supervised learning, using examples from a human play. This might give a more consistent result, and the agents might have been able to progress further into the levels. Then after the initial learning process, using reinforcement learning it could still learn how to play it differently or better after that.

Another relatively simple change for future work would be to exchange the target network architecture used for Q-learning with the dueling DQN model. This method should outperform the target network architecture, but was not considered until later on in the project, and it was decided that there was not

enough time to try to implement it.

Another structure that could be worth implementing would be convolutional neural networks [Krizhevsky et al., 2012] on top of the recurrent neural network. This could take a lot longer to train, which is why it was not chosen originally, but these kinds of networks are very good at finding patterns in images, and might have been able to improve the performance. With this it would also have the possibility to see different kinds of enemy types and learn how to handle different ones, instead of having all enemies be a single value. It could make it harder for it to learn where the player is though, as when changing suit, the color of the player also changes. This change could also work into one of the suggestions for implementing the powerups, mentioned earlier.

Keeping with the neural network theme, another possibility for future work would be to exchange the learning method entirely, and take more use of the case base, given that there are more cases than in the experiment in this thesis.

Instead of using reinforcement learning, it could use an algorithm like NEAT [Stanley and Miikkulainen, 2002], where it uses e.g. the N closest cases from the CBR part of the system and some randomly generated networks as the initial population, and evolve this to fit the case. This could give it yet another dimension, with neural network topologies tailored to a single case, rather than fit all the levels on a single topology.

It might also help in one of the problems presented in the discussion earlier, with a network in one case learning some behaviour that might be beneficial in several cases. If this method is used, this might be learned for agents in other levels as well, assuming it gets revisited during testing stages and still uses several cases from the case base.

Another possible solution to the problem of not distributing behaviours that benefit several cases could be to instead of having a case for every single possible case in the training space, it could only save the solution for cases as groups. If they are closer to each other than some set threshold, they all use the same solution. This could help against that problem, but could also limit the advantages of the CBR-ANN hybrid, especially for smaller case bases.

Yet another possible solution to this could be to use CBR-maintenance, and remove cases that are either not used much, or that performs poorly, and redo the training with the old unused and bad cases gone, making it possible for one case to inherit some behaviours from the better performing cases.

The last suggestion to this problem is to use multiagent systems, like mentioned in Chapter 4. Some way of the solutions of the cases to communicate and change each other would be very beneficial for the general level of the ANNs in the system. The previously mentioned NEAT implementation could be one such model.

A way to take more advantage of the CBR part of the system could be to

have several different neural network topologies for the different cases, other than to get this with NEAT. Different reward functions for different cases could be a possibility. This was not used in this experiment, but it could make it possible for a lot more advanced behaviour in each case, if each one of them, or each group of cases had a reward function that was tailored to them.

From this experiment, this could especially improve the performance in Quick man's and Crash man's stages, with giving rewards for going down and up respectively, while the other levels still only gets rewards for going right.

To make this model suitable for a continuous environment that cannot be paused, it would need to have a way of deciding when to switch to the CBR section of the program, from the ANN section of it. The experiment here used a hard limit on the time it had with each case before switching, both in training and testing. This would however not be a good solution for more "real" problems.

One way could be to switch when a case is determined to be terminal, that is, it is impossible to do anything more here, either the agent won, or it lost. Another solution could be to add another layer to the hybrid model, and having another agent learn when it's most valuable to switch between CBR and the ANN section of the system.

This could be trained with reinforcement learning while the program is running, or it could train on preset cases where it should switch modes with supervised learning, or it could possibly be trained with unsupervised learning.

Nothing like this was found in the literature, but is a very interesting idea, and could potentially make a very powerful system.

Other extensions for the CBR section of the system could be to swap out the similarity function with a more advanced method. One that seems very strong that could be seen in the Chapter 2, was to use neural networks in the retrieval process. This was the most common CBR-ANN hybrid in the literature, and did very well, giving hope that this might improve this model as well. Finally, it could also possibly use other methods than neural networks as the solutions for the different cases.

5.5 Conclusion

The thesis presents a new way of combining case-based reasoning and artificial neural networks to work in dynamic problems. Very little research on similar methods was found, and an experiment was therefore conducted to explore if this is a suitable method for dynamic environments. The experiment was to play the game Mega man 2, and the method was compared against the performance of a single ANN.

The CBR-ANN outperformed the single ANN in almost every case, suggesting that it is potentially a suitable method. It was concluded, however, that to

be able to say with confidence that it is a suitable solution, more research is required. The good performance compared to the single ANN answers the main research question with a very positive result, and indicates that it can be a suitable solution. However, the goal can not be said to have been reached with confidence, as the results still needs to be stronger before concluding.

The thesis contributes with a good foundation to work upon by presenting this new method for dynamic environments, as well as presenting suggestions for further research. These suggestions include improvements to the evidence as well as the method itself and the environment it is evaluated in.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

Asada, M. and Uchibe, E. (2001). Multiagent learning towards robocup. *New Generation Computing*, 19(2):103–120.

Bahar, M., Ghiasi, A., and Bahar, H. (2012). Grid roadmap based ann corridor search for collision free, path planning. *Scientia Iranica*, 19(6):1850–1855.

Biswas, S. K., Sinha, N., Purakayastha, B., and Marbaniang, L. (2014). Hybrid expert system using case based reasoning and neural network for classification. *Biologically Inspired Cognitive Architectures*, 9:57–70.

Bling, S. (2015). Mari/o - machine learning for video games. `https://www.youtube.com/watch?v=qv6UVOQ0F44`. [Online; accessed 25-February-2019].

Choy, K. L., Lee, W., and Lo, V. (2003). Design of an intelligent supplier relationship management system: a hybrid case based neural network approach. *Expert Systems with Applications*, 24(2):225–237.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

De Angelis, F., Boaro, M., Fuselli, D., Squartini, S., Piazza, F., and Wei, Q. (2013). Optimal home energy management under dynamic electrical and thermal constraints. *IEEE Transactions on Industrial Informatics*, 9(3):1518–1527.

De Paz, J. F., Bajo, J., González, A., Rodríguez, S., and Corchado, J. M. (2012). Combining case-based reasoning systems and support vector regression to evaluate the atmosphere–ocean interaction. *Knowledge and information systems*, 30(1):155–177.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer.

Galway, L., Charles, D., and Black, M. (2008). Machine learning in digital games: a survey. *Artificial Intelligence Review*, 29(2):123–161.

Garcia-Pardo, J. A., Soler, J., and Carrascosa, C. (2010). Social reinforcement learning for changing environments. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 269–272. IEEE.

Gonzalez, C., Lerch, J. F., and Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science*, 27(4):591–635.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Guo, Y., Hu, J., and Peng, Y. (2011). Research on cbr system based on data mining. *Applied Soft Computing*, 11(8):5006–5014.

Hegdal, S. S. and Kofod-Petersen, A. (2019). A cbr-ann hybrid for dynamic environments.

Henriet, J., Leni, P.-E., Laurent, R., Roxin, A., Chebel-Morello, B., Salomon, M., Farah, J., Broggio, D., Franck, D., and Makovicka, L. (2012). Adapting numerical representations of lung contours using case-based reasoning and artificial neural networks. In *International Conference on Case-Based Reasoning*, pages 137–151. Springer.

Herrero, Á., Navarro, M., Corchado, E., and Julián, V. (2013). Rt-movicabids: Addressing real-time intrusion detection. *Future Generation Computer Systems*, 29(1):250–261.

Heywood, M. I. (2015). Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines*, 16(3):283–326.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hong, J. and Prabhu, V. V. (2004). Distributed reinforcement learning control for batch sequencing and sizing in just-in-time manufacturing systems. *Applied Intelligence*, 20(1):71–87.

Horswill, I. (1995). Analysis of adaptation and environment. *Artificial Intelligence*, 73(1-2):1–30.

Huang, S.-Y., Yu, F., Tsaih, R.-H., and Huang, Y. (2015). Network-traffic anomaly detection with incremental majority learning. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE.

Kofod-Petersen, A. (2012). How to do a structured literature review in computer science. *Ver. 0.1. October*, 1.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32.

Liao, Z., Hannam, P. M., Xia, X., and Zhao, T. (2012). Integration of multi-technology on oil spill emergency preparedness. *Marine pollution bulletin*, 64(10):2117–2128.

Liu, Y.-c. et al. (2006). Hybridization of cbr and numeric soft computing techniques for mining of scarce construction databases. *Automation in Construction*, 15(1):33–46.

Lu, N., Lu, J., Zhang, G., and De Mantaras, R. L. (2016). A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, 230:108–133.

Ma, F., He, Y., Li, S., Chen, Y., and Liang, S. (2009). Research on case retrieval of case-based reasoning of motorcycle intelligent design. In *The Sixth International Symposium on Neural Networks (ISNN 2009)*, pages 759–768. Springer.

Mitchell, T. M. (1997). *Machine Learning*. WCB/McGraw-Hill.

Mitcher, M. and Weber, R. (2013). *Case-based reasoning: A Textbook*. Springer.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Morozs, N., Clarke, T., and Grace, D. (2016). Cognitive spectrum management in dynamic cellular environments: A case-based q-learning approach. *Engineering Applications of Artificial Intelligence*, 55:239–249.

Munoz, J., Gutierrez, G., and Sanchis, A. (2009). Controller for torcs created by imitation. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 271–278. IEEE.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Pinzón, C., De Paz, J. F., Bajo, J., Herrero, Á., and Corchado, E. (2010). Aiida-sql: an adaptive intelligent intrusion detector agent for detecting sql injection attacks. In *Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*, pages 73–78. IEEE.

Pinzón, C., de Paz, Y., Cano, R., and Rubio, M. P. (2009). An attack detection mechanism based on a distributed hierarchical multi-agent architecture for protecting databases. In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, pages 246–255. Springer.

Rahim, S. A., Yusof, A. M., and Bräunl, T. (2014). Genetically evolved action selection mechanism in a behavior-based system for target tracking. *Neurocomputing*, 133:84–94.

Reyes, E. R., Negny, S., Robles, G. C., and Le Lann, J. (2015). Improvement of online adaptation knowledge acquisition and reuse in case-based reasoning. *Application to process engineering design [J]. Engineering Applications of Artificial Intelligence*, 41:1–16.

Schlessinger, E., Bentley, P. J., and Lotto, R. B. (2006). Modular thinking: evolving modular neural networks for visual guidance of agents. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 215–222. ACM.

Schrum, J. and Miikkulainen, R. (2014). Evolving multimodal behavior with modular neural networks in ms. pac-man. In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pages 325–332. ACM.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

strategywiki.org (2017). Mega man 2/weapons. `https://strategywiki.org/wiki/Mega_Man_2/Weapons`. [Online; accessed 29-May-2019].

Sycara, K. P. (1990). Negotiation planning: An ai approach. *European Journal of Operational Research*, 46(2):216–234.

Tong, C. K., On, C. K., Teo, J., and Kiring, A. M. J. (2011). Evolving neural controllers using ga for warcraft 3-real time strategy game. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2011 Sixth International Conference on*, pages 15–20. IEEE.

Urdiales, C., Perez, E. J., Vázquez-Salceda, J., Sànchez-Marrè, M., and Sandoval, F. (2006). A purely reactive navigation scheme for dynamic environments using case-based reasoning. *Autonomous Robots*, 21(1):65–78.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Wu, Y., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Xu, B., Zhang, Y., Gong, D.-w., and Wang, L. (2017). A parallel multi-objective cooperative co-evolutionary algorithm with changing variables. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1888–1893. ACM.

Zehraoui, F., Kanawati, R., and Salotti, S. (2004). Casep2: Hybrid case-based reasoning system for sequence processing. In *European Conference on Case-Based Reasoning*, pages 449–463. Springer.

Appendix A

This appendix lists the hardware and software used to do the experiment of the thesis, and why if there were any options. It first lists the hardware, and follows up with the software.

5.6 Hardware used

Processor AMD Ryzen 7 1700X Eight-Core @ 3.5GHz

Graphics card Nvidia GeForce GTX 1080

Memory 16GB DDR4 @ 3200MHz

Motherboard Gigabyte Aorus GA-AX370-Gaming 5

5.7 Software used

Operating system Manjaro Linux 18.0.4 Illyria `https://manjaro.org/`

Kernel x86_64 Linux 5.0.18-1-MANJARO

C/C++ Compiler GCC version 8.3.0 `https://gcc.gnu.org/`

CUDA Cuda version 10.1.168-1 `https://developer.nvidia.com/cuda-zone`

Lua Lua version 5.1.5 `https://www.lua.org/`

Luajit Luajit version 2.0.4 `https://luajit.org/`

Torch Torch version 7 was used `http://torch.ch/`. Torch was chosen for the neural network part of the system over PyTorch, Theano and TensorFlow because of the emulator requiring lua. The following torch packages was used:

- RNN `https://github.com/Element-Research/rnn/`
- CUTORCH `https://github.com/torch/cutorch`
- CUNN `https://github.com/torch/cunn`
- Optim `https://github.com/torch/optim/`

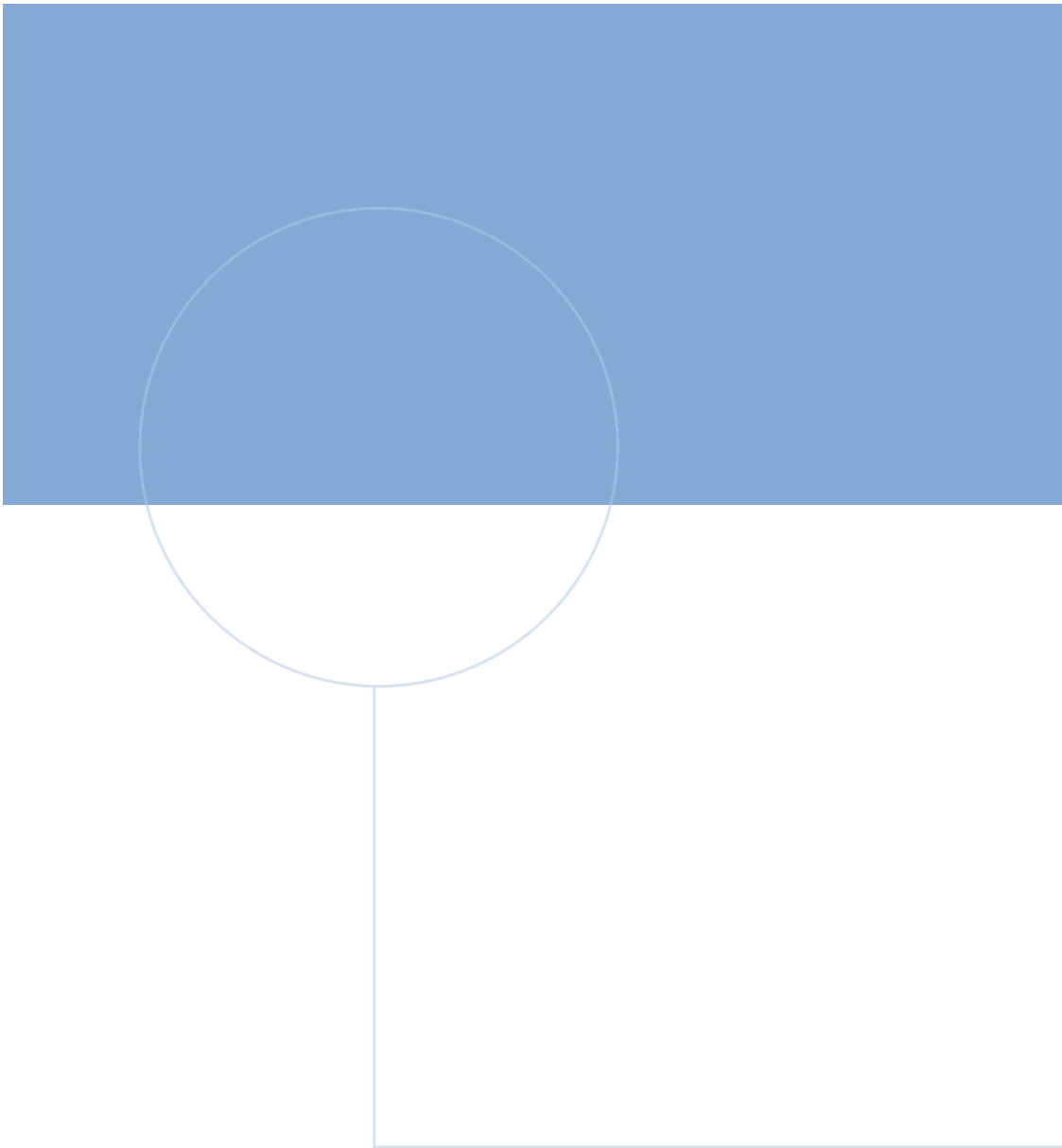
Lua packages Other lua packages used:

- socket `https://github.com/diegonehab/luasocket`
- json `https://github.com/rxi/json.lua`

CBR MyCBR `http://mycbr-project.org/` using the MyCBR rest api `https://github.com/ntnu-ai-lab/mycbr-rest` to communicate between the lua script and the MyCBR project. Only other alternative was jColibri. MyCBR was chosen above jColibri because of previous experience with it, the possibility of using the rest api, and more available help if needed at NTNU.

Emulator FCEUX version 2.2.3 debug `http://www.fceux.com/web/home.html`. Compiled with luajit rather than the built in lua version it comes with, to both be faster and for having torch available. Only other alternative that could emulate as well as FCEUX and have a possibility of scripting was the BizHawk emulator `http://tasvideos.org/BizHawk.html`. FCEUX was chosen because it could speed up the emulation more, speeding up the training process.

All links were verified February 6th 2020.



NTNU

Norwegian University of
Science and Technology