FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Combinatorial Solvers in Deep Reinforcement Learning
Student:	Richard Hájek
Supervisor:	Mgr. Radoslav Škoviera, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

The objective of this thesis will be to design, implement and debug a deep reinforcement learning (DRL) architecture in order to solve pathfinding and planning problems using combinatorial solvers. To enable the fusion of a neural network and a combinatorial solver, this thesis will be using Blackbox combinatorial solvers (described in this paper: https://openreview.net/pdf?id=BkevoJSYPB). Since the individual steps of combinatorial solvers (e.g. the A* pathfinding algorithm) typically do not allow differentiation, it is not possible to use them in a neural network with gradient descent optimization. The aforementioned paper proposes a solution that allows using error back propagation with these solvers, enabling their use as a layer of a standard neural network. Such combinatorial layer could be used in combination with the reinforcement learning paradigm to solve pathfinding and planning problems more efficiently than relying on pure neural network approach.
The task for the thesis will be to find a suitable toy problem and propose and design a solution for it using a blackbox combinatorial solver. The solution should be compared to a classical DRL approach. The baseline approach will be Q-Learning.

Tasks for the thesis:
1) Research literature for DRL and Combinatorial solvers.
2) Find a suitable pathfinding or planning problem that can be solved by DRL and develop a testing environment for it.
3) Propose a DRL algorithm that utilizes a blackbox combinatorial solver and implement it.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 20 September 2022 in Prague.

4) Compare the performance of the proposed method with basic Q-learning or other applicable methods.

Bachelor's thesis

COMBINATORIAL SOLVERS IN DEEP REINFORCEMENT LEARNING

Richard Hájek

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Radoslav Škoviera Ph.D.
May 11, 2023

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Hájek Richard. *Combinatorial Solvers in Deep Reinforcement Learning.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

List of code listings

I would like to thank everyone who supported me during my work on this thesis. I would like to thank my supervisor, Mgr. Radoslav Škoviera, Ph.D., whos advice proved to be invaluable and provided me with motivation troughout the thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2023 .....

Abstract

This bachelor's thesis measures the efficiency of alternative methods of modeling logical relationships between data in deep reinforcement learning. Leveraging SATNet [1], which can approximate these relationships, this bachelor's thesis has found that agents with a SAT solver are comparative in their results to their counterparts without the SAT solver.

Keywords MaxSAT, deep learning, reinforcement learning, parameter search, python, ENGLISH

Abstrakt

Tato bakalárska práce měří účinost alternatiního modelování logických vztahů mezi daty v rámci zpětnovazebního učení. Pomocí SATNetu, který dokáže tyto vztahy aproximovat, [1] bylo v bakalářské práci nalezeno, že agenti s SAT solverem v rámci zpětnovazebního učení podávají podobné výsledky jako agenti bez něho.

Klíčová slova MaxSAT, hluboké učení, posilovací učení, optimalizace parametrů, python, CZECH

Abbreviations and symbols

Abbreviations

ANN	Artificial Neural Network
NN	Neural Network
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
MAXSAT	Maximum Satisfiability
I	Total iterations
RLI	Check iterations
N	Total steps per iteration
BS	Replay buffer size
LS	Deep learning - started at step
TUI	Deep learning - target update interval
γ	Discount factor
τ	Soft update coefficient
TF	Train frequency
EF	Exploration factor - ratio of training where interpolated
EIE	Exploration factor - initial
EFE	Exploration factor - final
LR	Deep learning - learning rate
Batch S	Deep learning - batch size
Arch	Architecture of the policy
T	Training time in seconds
$\bar{R}$	Total return mean
σ_R	Total return standard deviation
FE	Feature Extractor
No-Op	No Operation or also called Identity function

Introduction

In modern neural networks, it is very difficult to model complex logical relationships. In order to approximate these, neural networks may use a disproportionate amount of computing power. As an alternative, we will explore the use of SATNet, which can model logical relationships in a single layer in an efficient way.

Therefore, we will evaluate a method of differentiating traditionally non-differentiable algorithms, such as SATNet, in the context of deep reinforcement learning. Due to the nature of deep learning, non-differentiable algorithms cannot be simply used, and other approaches are necessary.

We will evaluate the feasibility and viability of the SATNet deep neural network integrator [1] in the reinforcement learning environment. The original proposed solution has been to use Differentiation of Blackbox combinators (DFC)[2], however due to the following reasons SATNet has been ultimately chosen:

- The constraint satisfaction nature of the proposed environment has been deemed to be more suitable to SATNet than DFC
- Some parts of DFC require closed source 3rd party software, restricting adaptability of the code

As for deep Q learning, this thesis will attempt to find a suitable model:

$$model(obs, w) = action,$$

[calculation of output of the model, see chapter on deep learning]

which, when used, maximizes the return of the environment:

$$R = \sum_{n}^{0}(r_n),$$

[calculation of the total reward, see chapter on reinforcement learning]

of a learning environment for the reinforcement of hydraulic pipes. Agent will be rewarded by the following method:

- Penalty for any step taken
- Penalty for any state which is observed twice
- Reward for progress in any direction

The MaxSAT approach will be compared to deep Q-learning in order to estimate the viability of the method against industry standards.

The comparison will take into account the following:

- Total reward R
- Time taken T
- Steps taken S

Chapter 1

Deep Learning

This chapter introduces the topic of deep learning. The reader is encouraged to skip it if they are familiar with the topic.

1.1 Artificial Neural Networks

An artificial neural network, henceforth simply referred to as a neural network, or network, is a collection of mathematical computations organized in nodes. Nodes in the network represent neurons in a biological brain, and edges between nodes represent synapses. Computational nodes may henceforth be referred to as neurons or artificial neurons. The network has an input of arbitrary size and an output of arbitrary size [3]. The network may be thought of as a function accepting network inputs and network parameters resulting in some output.

$$model(x, w) = y$$

The computational functions in neural networks include large numbers of multiplications and sums. In addition to that, each neuron has an activation function which can be arbitrarily defined, as long as it is differentiable.

$$Y_{layer} = f_{activation}(X_{layer} * W_{layer})$$

Most common activation functions include rectified linear unit function, sigmoid, or tanh. The role of activation function in the neural network is to provide nonlinearity and thus to increase the complexity of the model. Activation functions can be defined over multiple neurons or over multiple data points in a batch, such as the softmax function or batch normalization. Artificial neurons may include stochastic behavior, such as dropout nodes. Artificial neural networks have two stages, training and inference. During training, the artificial neural network is optimized, that is, the parameter (w) of the model is being optimized. During inference, no changes are made to the network and only model output is calculated. Some nodes may behave differently during network training and inference. For example, batch normalization is typically a no-op during inference of the neural network.

Each network can be defined by a calculation graph and a set number of “parameters”. Parameters can be thought of as constant in network inference; their optimal value is what training hopes to find. Network may contain other variables, which may change during training; however, are not saved between inference or training, such as intermediary results in recurrent neural networks.

1.2 Supervised training

This thesis will primarily focus on reinforcement learning, which is based on supervised training. The objective of supervised training is to predict a set of output values given a set of input values. A collection of input-output pairs will henceforth be referred to as the ”dataset“. During training, each set of pairs is input into the network, and the network produces an output. The error can be computed as the difference between the expected output values and the inferred values. Minimizing this error is the goal of training the network, which makes the definition of the error function of paramount importance.

Each iteration of the training process is referred to as a batch. A batch consists of a set amount of training data. Larger batch sizes are preferred as they reduce the time needed to load the data into the RAM or GPU. However, batch sizes cannot be too large as this may exceed the available memory. A single training pass through the entire dataset is called an epoch. It may take many epochs to reach the optimal parameters of the model.

The training algorithm itself may require some parameters or configuration. To distinguish them from the learnable parameters and because they control the properties of the learnable parameters, they are called ”hyperparameters“. Selecting the appropriate hyperparameters is a nuanced process that may involve grid search, random search, other optimization techniques, or expert knowledge. Details of this process are outside the scope of this thesis, and this thesis shall use various forms of grid search, due to their extensibility and ease of use.

1.3 Gradient Descent

Gradient descent is an optimization algorithm used in deep learning to minimize the error of a model by updating its parameters iteratively. The algorithm starts with an initial set of parameters, then iteratively updates the parameters in the direction of the steepest reduction in the error until it reaches a minimum or a stopping criterion is met.

The basic idea behind gradient descent is to calculate the gradient of the cost function with respect to the parameters and then update the parameters in the direction of the negative gradient. The gradient represents the direction of maximum increase in the error, and hence updating the parameters in the opposite direction, i.e. the negative gradient, would lead to a reduction in the error.

One of the key hyperparameters in gradient descent is the learning rate, which determines the step size of the update. If the learning rate is too large, the algorithm may overshoot the minimum and may never converge, while if it is too small, the algorithm may converge too slowly.

To overcome the limitations of traditional gradient descent, more advanced optimization algorithms, such as RMSProp or Adam, have been developed. These algorithms allow for more granular control of the parameters' learning rate.

- STD (Stochastic Gradient Descent) is an algorithm for updating the w parameters of the neural network. It updates each w according to the following formula:

  $$w_i = w_i - \eta \nabla Q(w_i)$$

  Where w_i is a single parameter, η is the training rate and $\nabla Q(w_i)$ is the change of the resulting loss function with respect to this parameter.

- RMSprop (Root Mean Square Propagation) is a variant of gradient descent that uses a moving average of the squared gradient to scale the learning rate for each parameter.

  In this algorithm, one calculates the moving average:

  $$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

  Where γ is the "forgetting factor", a hyperparameter. And uses this to update the parameter w

  $$w = w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

- Adam (Adaptive Moment Estimation) is an optimization algorithm that computes adaptive learning rates for each parameter based on the previous gradient information. Adam is a 2014 improvement to the RMSProp algorithm, and its details are, due to their complexity, beyond the scope of this thesis. It has however been widely used since, due to its versatility.

The discussion of advantages and drawbacks of each choice of the training algorithm is also beyond the scope of this thesis. This thesis will use Ada, due to its adaptability and versatility.

Given the fundamental nature of gradient descent and related optimization algorithms, they are only applicable to differentiable functions. Nondifferentiable functions, such as the sign function, lack a gradient and cannot be used without being approximated. This thesis will focus on the examination and adoption of algorithms that are capable of addressing these limitations in the context of deep reinforcement learning.

Chapter 2

Artificial Neural Network Key Engineering Components

2.1 Artificial Neural Network

The artificial neural network is represented as a computational graph with parameters and inputs. The input and output size may not be known before inference or training. Any further mentions of “artificial neural network”, “neural network”, “network” or “model” refer to Artificial Neural Network.

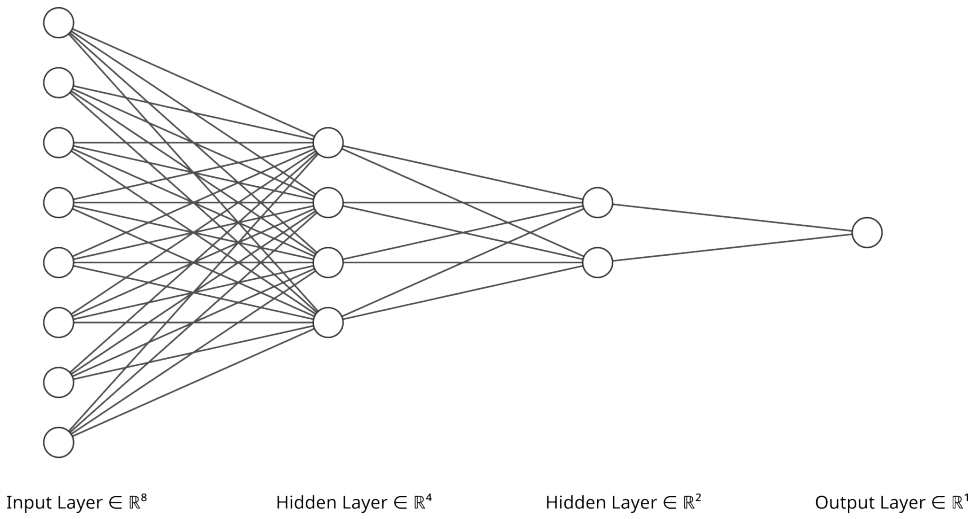

Figure 2.1 General structure of a deep neural network

This image shows an approximation of a deep neural network. There is an input layer, several hidden layers and one output layer with a single neuron.

2.2 Layers

The core of each neural network is a succession of layers [3]. A neural network may have an input layer and multiple output layers. Each of the layers computes its result with its input from the output of the previous layers. Each layer may have different properties, but each layer has some input size, some output size, and may have parameters, trainable and non-trainable.

This bachelor thesis will use the following layers:

- Linear layer: The output (before the activation function) is computed by matrix multiplication.
- Convolutional layer: The output is computed by a convolution over the input, with parameters of the convolution kernel trainable. This layer is widely used in image recognition, where context of each pixel is very important, whereas the absolute position of the pixel may not be.
- Dropout layer: During training, this layer randomly zeros some of the elements of the input with probability p using samples from a Bernoulli distribution. This behavior encourages the neural network to generalize. [4]

2.3 Activation Functions

Each layer in the neural network may be followed by an activation function. Examples of activation functions include:

- Linear activation: $f_{activation}(x) = x$
- ReLu activation: $f_{activation}(x) = \max(0, x)$ [5]
- Logistic function activation: $f_{activation}(x) = (1 + \exp(-x))^{-1}$
- SoftRelu: $f_{activation}(x) = \ln(1 + \exp(x))$

For appropriate activation function, neural network designer must consider its non-linearity and its derivability in expected ranges. It is inappropriate to use linear activation as activation function, as linear function is not nonlinear.

2.4 Loss Function

A loss function is an important component in training a model [6]. It measures the difference between the model's prediction and the true labels. The loss function may be in other works referred to as cost or objective function.

The loss function is used to evaluate the performance of the model in each iteration of the training process. The goal is to minimize the value of the loss function, which in turn means that the model predictions are getting closer to the true labels. The optimization algorithm afterwards uses the gradients of the loss function with respect to the model parameters to update the parameters in order to reduce the value of the

loss function. In this way, the loss function drives updates to the parameters in the entire network.

There are many examples of loss functions:

- Mean Squared Error (MSE): This is a popular loss function for regression problems where the goal is to predict a continuous value. MSE calculates the average of the squared differences between the model's predictions and the true labels. MSE is computaed as follows:

$$\mathcal{L} = (y - \hat{y})^2$$

- Binary Cross-Entropy (BCE): This is a loss function for binary classification problems, where the goal is to predict one of two possible outcomes. BCE measures the dissimilarity between predicted probabilities and true labels.

  BCE may be computed as:

$$\mathcal{L} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

  where y represents the true label (target) for a given sample. It is a binary value that is either 0 or 1. $\hat{y}$ represents the predicted output of the model for the given sample. It is also a binary value that is between 0 and 1, and represents the model's confidence that the sample belongs to the positive class (1) or negative class (0).

- Categorical Cross-Entropy (CCE): This is a loss function for multiclass classification problems, where the goal is to predict one of several possible outcomes. CCE measures the dissimilarity between the predicted probabilities and the true labels.

  CCE may be computed as:

$$\mathcal{L} = -\frac{1}{K} \sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

  where K is the number of classes, y_k is the true label (target) of the sample for the kth class (it is 1 if the sample belongs to class k and 0 otherwise), and $\hat{y}_k$ is the predicted probability of the sample belonging to the kth class.

- Hinge Loss: This is a loss function for binary classification problems, particularly for Support Vector Machines (SVMs). It measures the maximum margin between the model's predictions and the true labels.

  Hinge Loss may be computed as:

$$\mathcal{L} = \max(0, 1 - y \cdot \hat{y})$$

- Softmax Loss: This is a loss function for multiclass classification problems that is used in combination with the softmax activation function. It measures the dissimilarity between the predicted probabilities and the true labels.

  Softmax Loss may be computed as:

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

This thesis will use Mean Squared Error, due to it's versatility and ease of use.

Chapter 3

Reinforcement Learning

This chapter introduces the topic of reinforcement learning. The reader is encouraged to skip it if they are familiar with the topic.

Reinforcement learning is a type of machine learning that deals with the interaction between an agent and its environment. Agent learns how to behave in environment by performing actions and observing rewards it receive, like how animals and humans learn from experiences. This section focuses on core concepts of reinforcement learning, including reward signal, return, observation space, and action space, and their importance in the learning process. This chapter describes definitions as defined in [3]

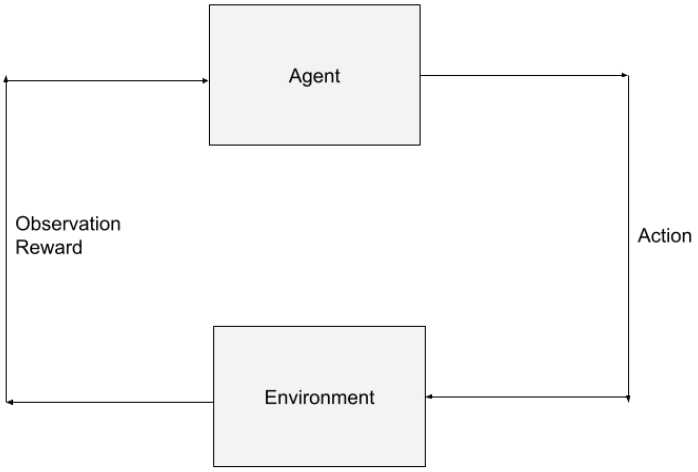

Figure 3.1 Reinforcement learning cycle

3.1 Core Concepts

3.1.1 Environment

The environment in reinforcement learning refers to the system or situation with which the agent interacts. It provides the agent with information about its state through observations, and the agent can take actions that affect the environment and receive rewards based on its actions. The environment represents the task or problem that the agent is trying to solve and it is through interaction with the environment that the agent learns to maximize its rewards. The environment can be a physical system, such as a robot navigating a maze, or a virtual system, such as a video game.

3.1.2 Reward

Reward value is the feedback agent receives after each action it takes. This value guides agent's learning by providing information about quality of actions it has taken. The ultimate goal of the agent is to maximize the total reward it receives over time, known as return.

3.1.3 Observation space

Observation space is set of all possible observations an observation agent can make about the environment. These observations provide the agent with information about the current state of environment, which is used to determine the next action. The action space is a set of all possible actions an agent can take.

3.1.4 Return

Agent's objective is to maximize total reward it receives, which is return. This is achieved by selecting actions that are likely to lead to high rewards in the future.

3.1.5 Policy

Agent may try to achieve the maximum return by using a policy, which maps from observation to, allegedly, the best actions. Policy is updated over time as the agent gains more information about environment and rewards it receives for its actions.

3.2 Categories

Reinforcement learning can be divided into several categories, depending on how the agent learns and updates its policy. These categories are on-policy reinforcement learning, off-policy reinforcement learning, model-based reinforcement learning, and model-free reinforcement learning.

3.2.1 On-policy

On-policy reinforcement learning is when an agent updates its policy based on the current policy. Agent's current policy is used to select actions that lead to rewards, and then this policy is updated based on rewards received. This is called "on-policy" because updates are made based on current policy. The advantage of on-policy reinforcement learning is that the agent always follows the most recent and accurate policy. The disadvantage is that it may introduce feedback loops that make training difficult.

3.2.2 Off-policy

Off-policy reinforcement learning is when the agent updates its policy based on a separate policy, which is called behavior policy. Behavior policy is used to select actions, and updates are made based on rewards received from environment. This is called an "off-policy" because updates are made based on a separate policy. The advantage of off-policy reinforcement learning is that the agent can learn from experience gathered by different policies.

3.2.3 Model-based

Model-based reinforcement learning is when an agent uses a model of environment to make predictions about the future. Model can be used to predict consequences of certain actions in environment and help agent to select best action to take. This is called "model-based" because the agent uses a model of environment to guide its learning. The advantage of model-based reinforcement learning is that the agent can learn faster because it has access to more information about the environment. The inherent disadvantage is that, in many problems, the environment cannot be accurately modeled.

3.2.4 Model-free

Model-free reinforcement learning is when the agent does not use a model of the environment. Instead, it updates the policy based on rewards received from environment without trying to understand the underlying mechanics of the environment. This is called "model-free" because the agent does not use a model of environment. The advantage of model-free reinforcement learning is that it is more straightforward and easier to implement, because the agent does not need to build and maintain a model of the environment.

This thesis will focus on on- and off-line policy reinforcement learning. This thesis will focus only on model-free algorithms.

Chapter 4

Algorithm Overview

4.1 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm that is widely used to solve problems in artificial intelligence and robotics. It is a value-based algorithm, which means it uses estimates of the expected future rewards of different actions to make decisions about which actions to take. [3] [7]

The core idea behind Q-Learning is to learn a function Q(s, a), which represents the expected future reward of taking action a in state s. [3] The algorithm starts with an initial estimate of the Q-function and then updates it over time through trial-and-error interactions with the environment. At each time step, the agent selects an action based on its current estimate of Q and then receives feedback in the form of a reward and the next state. The agent updates its estimate of Q based on this feedback, and the process repeats. The agent updates its estimate by the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

That entire process has been demonstrated in the following pseudo-code:

Code listing 4.1 Q-Learning

```
1. Initialize the Q-Table with zeros for all (state, action)
2. Repeat for each episode:
   a. Initialize the starting state
   b. Repeat for each step in the episode:
      i. Choose an action using an exploration strategy (e.g.
          epsilon-greedy)
      ii. Take the action and observe the next state and reward
      iii. Update the Q-Table using the Q-Learning equation:
           Q[state, action] = Q[state, action] + learning_rate *
               (reward + discount_factor * max(Q[next_state, :]) -
                Q[state, action])
      iv. Update the current state to be the next state
   c. Reduce the exploration rate (e.g. epsilon) for the next
```

```
        episode
```

One of the main advantages of Q-Learning is that it does not require a model of the environment's dynamics, which makes it well-suited to problems where the underlying system is unknown or difficult to model. Additionally, Q-Learning can handle problems with large or continuous state spaces, and it can also handle problems with nonstationary environments, where the rewards or transition dynamics change over time.

However, there are also some disadvantages to Q-Learning. One of the main limitations is that it requires a discrete action space, which means that the agent must be able to represent its possible actions as a finite set of discrete options. [3] This can make it difficult to apply Q-Learning to problems with continuous action spaces, such as robotic control or continuous control in video games. Additionally, Q-Learning can suffer from action overestimation, where the agent overestimates the expected future rewards of certain actions, leading it to make suboptimal decisions. [7] This can be mitigated through the use of function approximation techniques, such as deep neural networks, to represent the Q function.

4.2 Double Q Learning

Double Q-learning is a variation of the Q-learning algorithm. [7] But the main difference between double Q-learning and Q-learning is how they handle the estimation of the expected future rewards. In Q-learning, the algorithm uses only one Q-table to estimate the expected future reward for each action in a given state. This can lead to an overestimation of the expected reward for certain actions, causing the algorithm to make suboptimal decisions. Double Q-Learning solves this problem by using two Q-tables instead of one. The first Q-table is used to select the action to take, while the second Q-table is used to update the estimate of the expected reward for that action. This separation of the selection and update processes helps reduce the overestimation problem in Q-Learning. See the following pseudo-code for an implementation example:

Code listing 4.2 Double Q-Learning

```
1. Initialize two Q-Tables, Q1 and Q2
2. Repeat for each episode:
   a. Initialize the starting state
   b. Repeat for each step in the episode:
      i. Choose an action using an exploration strategy (e.g.
          epsilon-greedy)
         based on the sum of the Q-Tables:
           Qsum = Q1[state, :] + Q2[state, :]
           action = argmax(Qsum)
      ii. Take the action and observe the next state and reward
      iii. Update one of the Q-Tables using the Double Q-Learning
           equation:
           if rand() < 0.5:
               Q1[state, action] = Q1[state, action] +
                   learning_rate * (reward + discount_factor * Q2[
                   next_state, argmax(Q1[next_state, :])] - Q1[
```

```
                    state, action])
            else:
                Q2[state, action] = Q2[state, action] +
                    learning_rate * (reward + discount_factor * Q1[
                    next_state, argmax(Q2[next_state, :])] - Q2[
                    state, action])
        iv. Update the current state to be the next state
    c. Reduce the exploration rate (e.g. epsilon) for the next
        episode
```

Double Q-Learning is a model-free algorithm, which means that it does not require a model of the environment's dynamics. It also works well in problems with large or continuous state spaces, and non-stationary environments. Just like Q-learning, double Q-learning requires a discreet action space, and it can be slow to converge, especially in problems with large state spaces. [3] To address the slow convergence issue, researchers have developed variants of double Q-learning that incorporate exploration-exploitation trade-offs, such as epsilon-greedy exploration. [7]

4.3 Deep Q Learning

This algorithm utilizes a neural network, known as the **Q network**, to approximate the action value function, which represents the expected return of taking a specific action in a specific state. The Q network is trained by iteratively updating its parameters based on the temporal difference error, which measures the difference between the predicted and actual rewards received after taking an action.

The algorithm is called "deep" Q learning due to the utilization of deep neural networks, which have multiple hidden layers, as opposed to traditional Q learning algorithms that used shallow linear models or a table. The use of deep neural networks allows the algorithm to handle complex, high-dimensional state, making it applicable to a wide range of problems. [7]

In order to stabilize the learning process, the algorithm utilizes a technique called experience replay. Experience replay involves storing experiences, or transitions, in a memory buffer and randomly sampling mini-batches from this buffer to update the Q network. This allows the algorithm to learn from past experiences and avoid oscillations and divergences in the learning process. [7]

In **double deep Q learning**, another important component of deep Q learning is the use of a target network. The target network is a copy of the Q network that is used to estimate the expected return of taking a specific action in a specific state. The target network is updated less frequently than the Q network, which helps to reduce the variance in the estimate of the expected return. [7]

4.4 SAT

SAT is a problem in mathematical optimization that attempts to find whether or not a mathematical formula is satisfiable. The formula may be specified in different mathematical languages, such as first-order logic, other n-order logics or propositional logic.

For the purposes of this thesis, we will only consider propositional logic, which defines only infinite number of variables, $\wedge$ and $\vee$, meaning and and or respectively, $\neg$ representing logical not, and $=$ representing equality. Any formula may be written by any infinite number of equally valid representations, and for the purposes of SAT, the formula is converted to CNF, meaning conjunctive normal form. The formula in conjunctive normal form consists of several clauses, chained with $\wedge$s, where each clause is in turn composed of atoms chained with $\vee$s. [1]

4.5 MaxSAT

We may generalize the SAT problem to MaxSAT. MaxSAT problem over a CNF is defined as finding the maximum number of clauses which are satisfiable at the same time. MaxSAT is harder than SAT, and any solution to a MaxSAT problem over a CNF can be used to trivially find the solution to a SAT problem over the same CNF. Note that the opposite is not the case. [1]

4.6 SATNet

SATNet is an approximate solver for MaxSAT problems defined in CNF that is differentiable. The differentiability of this solver enables us to create a gradient throughout the solver. A SATNet layer uses a MAXSAT SDP relaxation with weights to generate informed predictions for the assignments of unknown variables using the discrete or probabilistic assignments of known MAXSAT variables as input. We analytically differentiate using the SDP relaxation to produce the backward pass. [1]

4.7 Supervised, unsupervised learning vs reinforcement learning

There are three general paradigms in machine learning: supervised, unsupervised, and reinforcement learning. These paradigms have different approaches to learning, the types of data they use, the models they employ, and how they are evaluated and optimized.

One major difference between reinforcement learning and the other two paradigms is their approach to learning. Reinforcement learning works by trial and error, where an agent interacts with the environment and receives rewards or punishments as feedback. On the other hand, supervised or unsupervised learning trains on labeled or unlabeled data without any external feedback. [3]

Another difference is the type of data for which these techniques are used. Reinforcement learning is typically applied for sequential decision making, where the agent takes actions over time and receives feedback at each time step. Supervised and unsupervised paradigms are typically applied to static data like images, video, and text to recognize complex patterns and relationships in the data.

The models used in reinforcement learning and deep learning are also different. In reinforcement learning, the agent's policy is represented by a possibly stochastic function that maps states to actions. The aim is to learn an optimal policy that maximizes the

cumulative reward over time. In contrast, supervised learning models are typically feedforward or recurrent neural networks that map inputs vectors to some labels (self-discovered lables in case of unsupervised learning).

In addition, the evaluation and optimization of reinforcement learning and the other paradigms also differ. Reinforcement learning models are evaluated based on their long-term performance, such as total reward over a period of time. The goal is to optimize the agent's policy to maximize this long-term performance. The other paradigms, on the other hand, try to minimize some objective function (e.g., loss for supervised learning).

Chapter 5

Environment to solve

5.1 Hydraulic environment with rotation control

This environment represents a popular arcade game colloquially referred to as ”Water Pipe”, ”Plumber game” or ”Pipes”. The goal of this environment is to rotate a series of pipes in order to allow water to flow from a singular inlet to a singular outlet. We implemented the environment in a popular OpenAI Gym framework, which facilitates communication between the environment and reinforcement learning algorithms [8].

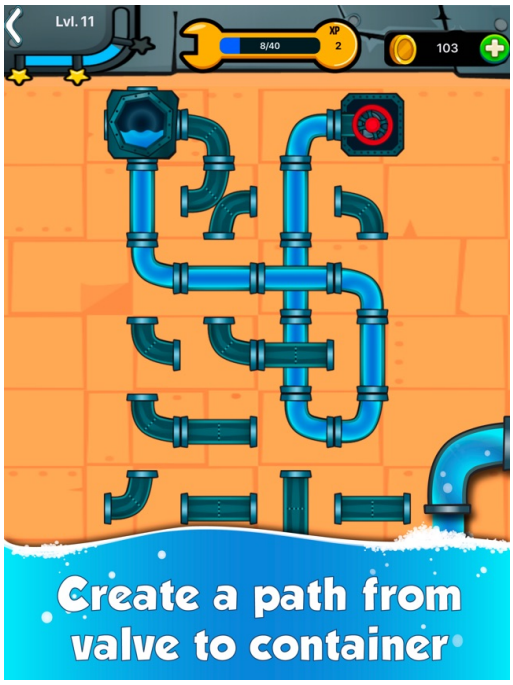

Figure 5.1 Illustrative image - Water pipes : pipeline game [9]

An illustrative image, taken from the Apple App Store, which shows a kids game where the player must connect pipes from water source to target. We reimplemented this environment in Python to allow for development of reinforcement learning algoithms.

5.1.1 State space

This environment consist of a $n \times n$ grid, on each grid any of the following tiles may be represented:

- Empty tile
- Straight pipe
- Angled pipe
- T-pipe
- 4 way pipe

In addition, each tile may be rotated in any direction of $\uparrow, \leftarrow, \downarrow, \rightarrow$.
The state space can therefore be defined with two matricies as follows:

$$S_{tile,i,j} \in \{empty, pipe_{straight}, pipe_{angle}, pipe_{T}, pipe_{fourway}\} \mid 0 \leq i, j < n$$

$$S_{angle,i,j} \in \{0, 1, 2, 3\} \mid 0 \leq i, j < n$$

5.1.2 Observation space

This environment is fully observable. The observation space is identical to the state space and thus the agent can observe the full environment at any point in time.

5.1.3 Action Space

The agent can turn any tile on the grid by 90deg.

$$A_{i,j} \in \{0, 1\}^{n \times n}$$

5.1.4 Reward

Agent receives -1 reward for all turns in which the environment is not solved. Agent additionally receives -100 for each time the agent finds itself in the same observation as before. Agent receives positive +N reward for all N pipes which were flooded which weren't flooded before.

5.1.5 Examples

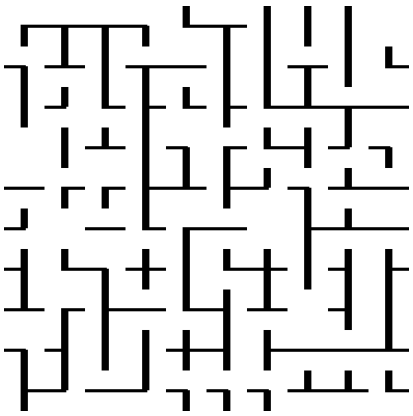

Figure 5.2 Implemented environment - shuffled

This image shows the implemented environment in a shuffled state - there are no active flows of water. The agent must rotate each pipe to direct the flow of water.

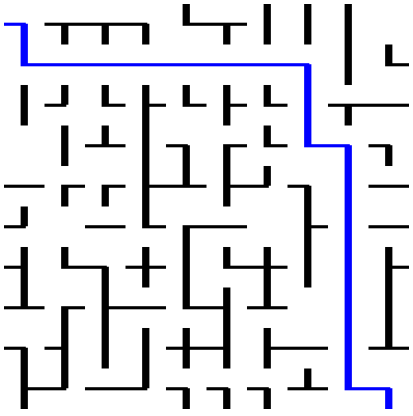

Figure 5.3 Implemented environment - solved

This image shows the implemented environment in a solved state - there is a water flow from the inlet (always top left) to the outlet (always bottom right).

Chapter 6

Agent Architectures

6.1 Deep Double Q Learning Agent (DQN)

The agent used will be a double Q learning algorithm with deep learning as a Q-table back-end. Therefore, this agent can deal with a large variety of environments including those defined in the previous chapter. The advantages of this agent is high flexibility, as this agent can adapt to virtually any input type and shape. The disadvantages are that the output still has to be a discrete action. Fix for this is so called actor-critic architecture, which is however outside of the scope of this thesis.

Ultimately, the DQN agent has been chosen because it is the most efficient and is applicable to this problem.

6.1.1 Agent Architecture

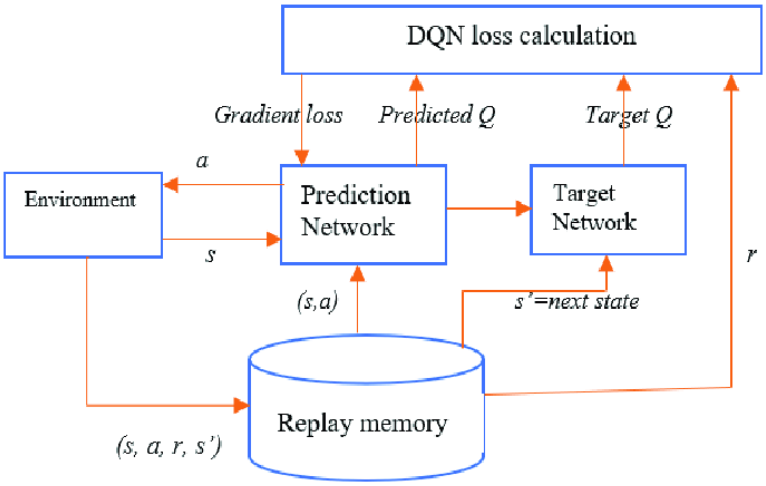

Figure 6.1 DQN Agent overview [10]

This figure shows the data flow of the implemented DQN agent. The core is the replay buffer which stores past experiences.

This agent contains 3 main components, the replay buffer, the learning algorithm, and the Q-table backend. All of these have their own hyperparameters. Unless stated otherwise, none of the hyperparameters are tuned during the training.

The replay buffer are the memories of the agent. The agent puts all experiences in this ring buffer and forgets the ones that are too old. The replay buffer has some maximum size, and when its full some memories need to be dropped. In the architecture of this thesis, the oldest memories are dropped. There are various names for each of these in the literature; replay buffer may be referred to as just buffer, or more poetically, memory buffer.

The learning algorithm is double Q learning, as discussed in earlier chapters. This algorithm is resistant towards noise feedback loops and is suitable for the environments proposed in this thesis.

The Q-table backend is the estimator of the Q-values. The agent uses various algorithms to improve the estimator and, in turn, achieve more accurate Q-values.

6.1.2 Learning steps

The learning of this agent must balance several steps:

- Exploration of the environment
- Training the estimator
- Evaluating the agent on the environment

Each of these steps has many hyperparameters, use of which we shall discuss in the following sections.

6.1.3 Exploration

To collect memories, the agent has to explore an environment. Each step, the agent may either take a random step or a step as recommended by the Q-table. The hyperparameter `exploration_factor` controls the ratio of random steps vs recommended steps. Too high a ratio causes the agent to take too many random steps, effectively slowing down learning in environments sensitive to small mistakes. However, a small exploration rate causes the agent to decrease the exploration of the environment and limits its experience. This hyperparameter starts with a large value, such as 0.5 and may be lowered over time.

The hyperparameter `buffer_size` controls the maximum size of the replay buffer. The optimum value chosen for this thesis is in the range of 10,000. This ensures big enough dataset which takes less than 1 minute to create, allowing for rapid prototyping.

6.1.4 Training the estimator

The estimator must be trained to improve the prediction of the Q-table to enable the agent to make better choices. Training is run on the memory replay buffer as a dataset. However, the memory replay buffer contains raw experiences and cannot be used to

train the estimator directly. First, we must calculate the estimated Q values from the memories using the current estimator.

We have learned, that this introduces a vicious feedback loop, where a bad estimator causes bad training data. Double Q learning addresses this issue to a certain extent.

The hyperparameters at play here are `max_dataset_size`, `copy_to_target_rate`, `learning_rate` and `discount_factor`. This is in addition to the estimators own hyperparameters, such as `learning_rate`, a different one though, or `batch_size`. The most important hyperparameter is `discount_factor`, which controls how much the agent "consideres" the future when calculating the Q-values. Especially in the beginning, the `discount_factor` should be fairly low, as the estimators are unrealiable anyway and its important to limit the error feedback loop. The `*_rate` hyperparameters control how often the estimator is trained, and its tuning mainly affects the speed of training, not the performance.

6.1.5 Deep Neural Network architecture

In the deep neural network used in the agent has the following architecture.

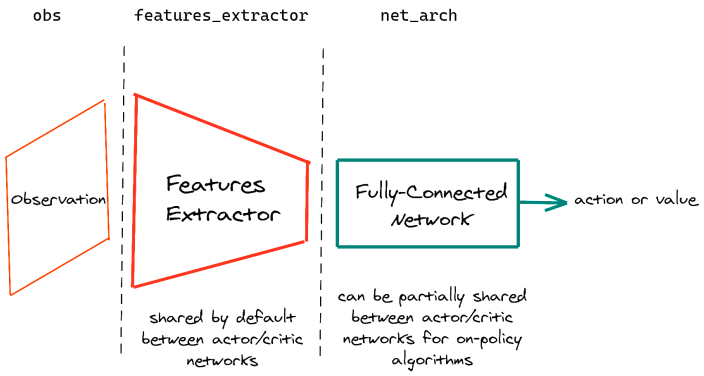

Figure 6.2 Netwrk architecture [11]
This is the architecture of the Q-network, which is inside the prediction network in 6.1. The SatNET layer will be included in the feature extractor part of the network for the SATNet agent.

Observation is taken directly from the environment. The neural network is divided into two sections, feature extractor and a fully connected network. It is divided this way to simplify learning when using other algorithms, which are not relevant in this thesis.

6.1.6 Evaluating the agent

It is important to know how well the agent is doing, which may be difficult to grasp, due to the `exploration_factor`. Therefore, every now and then it is important to evaluate the agent without randomness and estimate the agent's performance.

6.2 Double Q Learning SATNet agent

The double Q learning agent contains a MaxSAT layer in the feature extractor. MaxSAT layer also has a fully connected layer preceding it, to help to decrease the dimensionality of the input.

Chapter 7

Results

In order to establish a reliable comparison between the two methods, a parameter search was conducted, with repeated runs for each set of parameters and repeated evaluations of the final agents. For each parameter set, the agent has been trained 5 times and evaluated on at least 10 episodes, which gives the results statistical significance. The goal is considered met when $\bar{R}$ is higher than -20.

We can observe 7.1 that the DQN agent was able to solve the environment several times. The MaxSAT powered networks have proven themselves to reach the satisfactory results at roughly the same rate. 7.2

An interested reader might find the source code to repeat the experiments here `https://github.com/richard-hajek/bachelors-thesis/`

	EF	EIE	EFE	LR	Batch S	Arch	$\bar{R}$	σ_R	T
0	0.5	1	0.05	0.001	64	[9]	-121	95	292
1	0.5	1	0.05	0.0001	64	[9]	-140	90	281
2	0.5	1	0.05	1e-05	64	[9]	-180	59	261
3	0.5	1	0.05	0.001	64	[18]	-42	78	318
4	0.5	1	0.05	0.0001	64	[18]	-4	2	310
5	0.5	1	0.05	1e-05	64	[18]	-180	59	251
6	0.8	1	0.05	0.001	64	[9]	-22	59	252
7	0.8	1	0.05	0.0001	64	[9]	-81	96	245
8	0.8	1	0.05	1e-05	64	[9]	-140	91	258
9	0.8	1	0.05	0.001	64	[18]	-2	1	322
10	0.8	1	0.05	0.0001	64	[18]	-23	58	272
11	0.8	1	0.05	1e-05	64	[18]	-160	79	259

Table 7.1 DQN Agent testing results on the environment

This table shows the results of parameter search of the DQN agent on the implemented environment. It shows that for a small amount of the searched parameters, the agent has successfully adapted to the environment and is able to solve it above the threshold considered success. For the explanation of the variables, see the abbreviations table.

	EF	EIE	EFE	LR	Batch S	FE	Arch	$\bar{R}$	σ_R	T
0	0.5	1	0.05	0.001	64	SATNet	[25]	-200	0	350
1	0.5	1	0.05	1e-05	64	SATNet	[25]	-66	47	630
2	0.5	1	0.05	0.001	64	SATNet	[9]	-16	11	842
3	0.5	1	0.05	0.0001	64	SATNet	[9]	-21	15	822
4	0.5	1	0.05	1e-05	64	SATNet	[9]	-14	25	642
5	0.8	1	0.05	1e-05	64	SATNet	[9]	-47	52	774
6	0.5	1	0.05	0.0001	64	SATNet	[25]	-50	65	152
7	0.8	1	0.05	0.001	64	SATNet	[25]	-79	34	485
8	0.8	1	0.05	0.0001	64	SATNet	[25]	-15	11	468
9	0.8	1	0.05	1e-05	64	SATNet	[25]	-200	0	436
10	0.8	1	0.05	0.001	64	SATNet	[9]	-22	10	292
11	0.8	1	0.05	0.0001	64	SATNet	[9]	-112	80	558

Table 7.2 SATNet results

This table shows results from the parameters search with the SATNet network included in the agent. The agent was capable of reaching the goal state as did the Q-Learning method, however did not substantially improve. We also did these tests on a significantly more powerful machine than the classical DQN ones.

	Parameter	Constant Value
0	Learning iterations	10
1	Check iterations	5
2	Deep learning - started at step	5000
3	Deep learning - target update interval	1000
4	Train frequency	4
5	Total steps per learning iteration	20000
6	Replay buffer size	100000
7	Soft update coefficient τ	1
8	Discount factor γ	1
9	Neural network backend	MlpPolicy

Table 7.3 Constant Parameters

These parameters remained unchanged throughout the training process. Each parameter was tested on Learning iterations $*$ Check iterations $*$ Total steps per learning iteration steps.

Chapter 8

Conclusion

We have implemented a discrete-logic focused reinforcement learning environment in the popular framework Gym.ai, 5.1. This environment is replicating an arcade game, where a flow of water must be established from a source location to a target location. For the examples of the environment see figures 5.2, 5.3.

We have compared the results of the classical DQN and of DQN enhanced by a MaxSAT solver. We have compared them on metrics such as speed of training, mean return and the deviation of the mean return.

The classical DQN agent was used as a baseline and showed some satisfactory results 7.1, however with mean of -90 and standard deviation of -63. The agent which has been enhanced by the MaxSAT solver showed viable progress 7.2, however unfortunately, it did not show substantial gains over the classical DQN agent, in addition to taking significantly more time to train. The MaxSAT agent showed mean of -70 with standard deviation of around -30.

We have concluded however that any gains are not substantial enough and would require more experimentation.

Bibliography

1. WANG, Po-Wei; DONTI, Priya L.; WILDER, Bryan; KOLTER, Zico. *SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1905.12149.
2. VLASTELICA, Marin; PAULUS, Anselm; MUSIL, Vít; MARTIUS, Georg; ROLÍNEK, Michal. *Differentiation of Blackbox Combinatorial Solvers*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1912.02175.
3. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: a modern approach*. 3rd ed. Pearson, 2009.
4. LABACH, Alex; SALEHINEJAD, Hojjat; VALAEE, Shahrokh. *Survey of Dropout Methods for Deep Neural Networks*. 2019. Available from arXiv: 1904.13310 [cs.NE].
5. AGARAP, Abien Fred. *Deep Learning using Rectified Linear Units (ReLU)*. 2019. Available from arXiv: 1803.08375 [cs.NE].
6. DRÄGER, Simon; DUNKELAU, Jannik. *Evaluating the Impact of Loss Function Variation in Deep Learning for Classification*. 2022. Available from arXiv: 2210.16003 [cs.LG].
7. HESSEL, Matteo; MODAYIL, Joseph; HASSELT, Hado van; SCHAUL, Tom; OSTROVSKI, Georg; DABNEY, Will; HORGAN, Dan; PIOT, Bilal; AZAR, Mohammad; SILVER, David. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. Available from arXiv: 1710.02298 [cs.AI].
8. BROCKMAN, Greg; CHEUNG, Vicki; PETTERSSON, Ludwig; SCHNEIDER, Jonas; SCHULMAN, John; TANG, Jie; ZAREMBA, Wojciech. *OpenAI Gym*. 2016. Available from eprint: arXiv:1606.01540.
9. *Water pipes : pipeline — apps.apple.com* [https://apps.apple.com/us/app/water-pipes-pipeline/id1341680606]. [N.d.]. [Accessed 09-May-2023].
10. ARWA, Erick; FOLLY, Komla. Reinforcement Learning Techniques for Optimal Power Control in Grid-Connected Microgrids: A Comprehensive Review. *IEEE Access*. 2020, vol. 8, pp. 1–16. Available from DOI: 10.1109/ACCESS.2020.3038735.

11. RAFFIN, Antonin; HILL, Ashley; GLEAVE, Adam; KANERVISTO, Anssi; ERNESTUS, Maximilian; DORMANN, Noah. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*. 2021, vol. 22, no. 268, pp. 1–8. Available also from: `http://jmlr.org/papers/v22/20-1364.html`.