Automated design of distillation sequences within Aspen using Reinforcement Learning.

Department of Sustainable Process Technology

Faculty: Science and Technology

BSc Thesis in Chemical Science and Engineering

Written by Richard ten Hagen

Under the Supervision of Prof. Dr. Ing. Meik B. Franke

Groups: SPT and PDO

Year 2020-2021

Table of Contents

[Nomenclature 5](#_Toc75695533)

[List of Abbreviations 6](#_Toc75695534)

[1. Introduction 7](#_Toc75695535)

[1.1 Problem Introduktion 7](#_Toc75695536)

[1.2. Historical background 7](#_Toc75695537)

[1.2.1. Chemical industry 7](#_Toc75695538)

[1.2.2. Computer aided learning 8](#_Toc75695539)

[1.3. Research questions 9](#_Toc75695540)

[1.4. Aims and Objectives 9](#_Toc75695541)

[1.5. Reinforcement learning (RL) 10](#_Toc75695542)

[1.5.1. History of RL 10](#_Toc75695543)

[1.5.2. Markov process 10](#_Toc75695544)

[1.5.3. Components of RL 10](#_Toc75695545)

[*1.5.3.1. Agent* 11](#_Toc75695546)

[1.6. Deep Q network (DQN) 12](#_Toc75695547)

[2. Literature research 15](#_Toc75695548)

[2.1. Case study 15](#_Toc75695549)

[2.2. Search Space 15](#_Toc75695550)

[2.2.1. Adding more parameters 16](#_Toc75695551)

[2.2.2. Runtime length 16](#_Toc75695552)

[2.3. Algorithms for Space search 16](#_Toc75695553)

[2.3.1. Heuristics and Rule based limitations: 16](#_Toc75695554)

[2.3.2. Branch and Bound 17](#_Toc75695555)

[2.3.3. Gradient accent 17](#_Toc75695556)

[2.3.4. Evolutionary strategy 17](#_Toc75695557)

[2.4. Machine learning 18](#_Toc75695558)

[2.4.1. Supervised learning 18](#_Toc75695559)

[2.4.2. Unsupervised learning 18](#_Toc75695560)

[2.4.3. Reinforced learning 19](#_Toc75695561)

[2.5. Reward function 19](#_Toc75695562)

[2.5.1. Income 20](#_Toc75695563)

[2.5.2. Expense 20](#_Toc75695564)

[2.5.3. Column capital costing 20](#_Toc75695565)

[2.5.4. Total Annual Cost (TAC) 20](#_Toc75695566)

[2.5.5. Punishments 21](#_Toc75695567)

[4. Methodology 22](#_Toc75695568)

[4.1. Notations 22](#_Toc75695569)

[4.2. ASPEN DSTWU environment 22](#_Toc75695570)

[4.3. Pseudo code 23](#_Toc75695571)

[5. Results 24](#_Toc75695572)

[5.1. Agent evaluation metrics 24](#_Toc75695573)

[5.2. Changes to the model 25](#_Toc75695574)

[5.2.1. Change the value of the compounds 25](#_Toc75695575)

[5.2.2. Hard maximum reward to limit the minimal reward 26](#_Toc75695576)

[5.2.3. Simple DQN 26](#_Toc75695577)

[5.3. Punishing results in adaptation 26](#_Toc75695578)

[5.4. Agent makes use of errors to manipulate reward function 27](#_Toc75695579)

[5.5. Heuristics vs best agent choice 27](#_Toc75695580)

[5.5.1. Heuristically solutions to sequencing 28](#_Toc75695581)

[5.5.2. Best Action choices 28](#_Toc75695582)

[5.5.3. Comparison between heuristic and agent 29](#_Toc75695583)

[5.6. High vs low exploration 29](#_Toc75695584)

[5.6.1. Comparison to random choices 30](#_Toc75695585)

[5.7. Runtime analysis 31](#_Toc75695586)

[6. Conclusions 32](#_Toc75695587)

[6.1. Answering Research questions 32](#_Toc75695588)

[6.2. Problems with the methodology 32](#_Toc75695589)

[6.2.1. Runtime 32](#_Toc75695590)

[6.2.2. Stability 33](#_Toc75695591)

[6.2.3. Custom DQN program 33](#_Toc75695592)

[6.2.4. Mistakes 33](#_Toc75695593)

[6.3. Outlook 33](#_Toc75695594)

[6.3.1. Learning parameter optimization: 33](#_Toc75695595)

[6.3.2. Comparison to other algorithms 34](#_Toc75695596)

[6.3.3. Larger set of training 34](#_Toc75695597)

[7. APPENDIX 34](#_Toc75695598)

[References 34](#_Toc75695599)

[Presentation 36](#_Toc75695600)

# Nomenclature

|  |  |
| --- | --- |
| Variables | Definition |
| S | Space of possible states, if it’s a specific case its ‘’s’’ and s’ being the future one |
| St | State space in dependence of time |
| A | Space of all possible actions, specific action is “a” |
| T | Space of all transformation which can convert one state to another |
| R | Space of Rewards where each T is assigned a scalar value “r” |
| 𝑃[𝑆𝑡 + 1|𝑆𝑡] | Probability of going from St to S where t is one larger |
|  | Policy which is reward prediction for any action in any state |
| Q(𝑠, 𝑎) | Q value is a reward which is found by the Q Bell equation with future reward |
| 𝛾 | Discount factor for future reward |
|  | Node activation, scalar value which represents nodal importance |
|  | Weight factor for transferring nodal activation to next node |
|  | Bias, minimum activation energy for nodal transfer |
|  | Vector of Nodal activations in layer n |
|  | Matrix of all weight factors between two layers |
|  | Vector of all Biases |
|  | Minimum Reflux ratio |

# List of Abbreviations

|  |  |
| --- | --- |
| Term | Definition |
| ML | Machine learning |
| RL | Reinforcement learning |
| DC | Distillation columns |
| NN | Neural net |
| DNN | Deep Neural net |
| MINLP | Mixed Integer Non linear problem |
| DQN | Deep Q network |
| DSTWU | Name of Aspen distillation shortcut |
| MDP | Markov decision process |
| Q table | Tabular list of the reward for each action-state pair |
| TAC | Total annual cost |

**Acknowledgements:**

First and foremost, I would like to thank my parents who raised me and trained my neural network to be capable of learning how to train a neural network. Besides that I would like to thank Christos Abatzis for being a good friend, tolerating my annoyingness and aiding me in my work with Aspen. Laurence Midgley was a great inspiration and a seemingly endless well of knowledge for all questions concerning artificial intelligence and machine learning without whom this project would have been impossible. My daily supervisor Meik Franke helped me with inspiration and experience necessary for completing the labyrinth of academic standards. Since my writing tends to be incomprehensible for humans, I also extend my heartfelt gratitude to Alexander, Friedrich, Johannes, Mama and Chris for proofreading my paper.

***Abstract:***

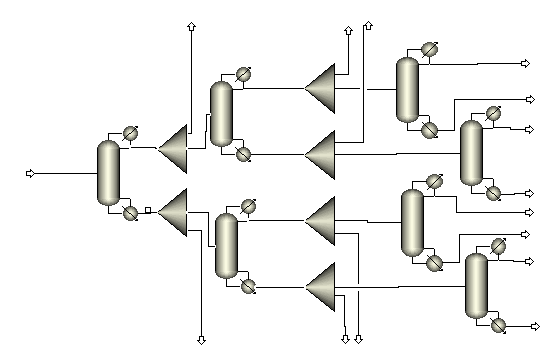
This papers research is a continuation of previous work by Laurance Midgley containing to RL usage for the sequence determination of distillation columns, with the added components of integrating Aspen Plus as the simulation program. The Winn-Underwood-Gilliland method is used for the simulation which allows for a capital cost, operational cost and purity calculation with Keras TensorFlow being the DQN training platform. A very generalized approach was used where a minimal amount of human input was hard coded. This enables the methodology to be utilized for other problems but with the current training time it was not possible to reach stable convergence of the policy function for the Case example. As a result, various sequences are proposed for further analysis and optimization and compared to heuristic solutions. The neural network was still able to prove its capability of learning to mitigate punishment as well as preventing errors. There are also propositions for the value of various variables given which have been found to be ideal for the training in the context of chemical engineering.

# 1. Introduction

## 1.1 Problem Introduction

One of the problems which chemical engineers face is the upscaling of known small-scale experimental methods to larger scales. One of the tools used here is process design. Since there are high costs in manufacturing a pilot plant it is necessary to simulate as much of the process as possible before ever even building it. This process usually involves highly paid labour such as chemical engineers developing the process in detail before it can be implemented. If the costs of producing these detailed plans could be lowered the innovation rate would be much higher while maintaining the costs. This is no new idea and there have been various methodologies (Venkatasubramani) implemented with this goal in mind.

The design of chemical processes is based on robust physical models as well as methodologies which allow for the efficient design of a system which produces a product. These include generating detailed documentations about what requirements and constraints are set and how production could be possible under these conditions. Some of these steps such as synthesis design, process flow sheets and design of unit operations have already been automated in software such as Aspen Plus, but there is still the need for human interaction in order to provide the software with the capability of human intuition and common sense.

For optimization of a system the currently most common method is mixed integer non-linear programming (MINLP) (Yixin Ye) which requires all possible configurations to be predefined and embedded in one system where the optimal solutions are searched for and approximated.

In Figure 1 is a process example shown where various possibilities are given in one structure. By choosing the split factor to be zero or 1, it is possible to make the choice between using the stream as a product or doing further distillation. By optimizing this structure, it is possible to find the sequence which is best for the given goals from the predefined list.

This means that this type of optimizing cannot be applied to open-ended questions since all cases need to be predefined. The design of distillation systems is inherently open-ended since it is always possible to add another distillation column to the branches – and this means, that MINLP has only limited applicability. These limitations will become more apparent in the chapter 2.1 about spaces and reductions.

Figure 1 superstructure approach to Process design

The proposed methodology in this assignment will attempt to solve these limitations by utilizing a neural network learning algorithm to automatically generate distillation column design and finding optimal solutions for a given feed.

## 1.2. Historical background

### 1.2.1. Chemical industry

The word ‘’chemistry’’ has a rich history which can trace its etymological roots all the way back to antiquity in either Greece or Egypt (Liddell).But the industrial production of chemicals is far older. Clay tablets from Mesopotamia 5000 years ago showing the potassium extracted from potash for soap production and the usage of quicklime (Carran) dates back at least twice as far. The production of alkaline substances is not the only chemistry (Seth) but the mixing of plant juices and alloying metals was dominating the craft, which over time developed to alchemy. This extraction of natural substances continued until the development of synthetic chemistry in the last century (Scott D. Barnicki), which allowed for the creation of molecules, which previously did not exist in nature. These developments were combined with the start of mass production for steel and coal products and the subsequent production of natural compounds like salicylic acid through synthetic processes which resulted in aspirin. With the shift in attention to polymers and plastic in the 1930s (Scott D. Barnicki) spectacular growth occurred with the utilization of petroleum and this matured the field of unit operations with the development of heuristics and design analogies. In the last half century this growth slowed down since plastic products saturated the market, which resulted in more competitive pricing. Globalization showed a path out of this dilemma by using the upcoming markets in developing countries as importers of products thanks to lower trade barriers. Where transport was not viable multinational companies were building plants overseas in order to circumvent shipping costs. This resulted in the current situation of worldwide competition in a growing market. This will mandate the necessity for plant designs to work with smaller margins and flexible production. The now all-encompassing topic of climate change will make it necessary to lower the carbon production and thus redesign many processes.

Therefore (Scott D. Barnicki) it might be necessary to reverse the previous differentiation into unit operations and to start optimizing the plant holistically. With equilibrium-based operations having reached a high level of sophistication there is still potential in rate defined reactions, which are much less used in the current industry such as permeation, chromatography, ion exchange or electrophoresis. All these problems necessitate that the methods used in redesigning chemical plants are improved to enable the changes, which will be required. Utilizing computer aided design will be part of this process. The capability of machines to analyse large quantities of data very rapidly at a low cost offers the possibility of redesigning processes where humans would not have had the time. Especially the area of machine learning (ML), where a computer retrieves knowledge about the world and utilizes it, is one of the most obvious implementations.

### 1.2.2. Computer aided learning

The historical implementation of ML (Venkatasubramani) systems in the field of chemical engineering is divided into two approaches: the expert systems which are “classical” computer programs which have some knowledge and rules about the world coded into their memory, which allows them to make inferences about the problem at hand and the Neural Networks (NN). An example (Venkatasubramani) for this is CONPHYDE (1983) which predicted properties of complex fluid mixtures or DECADE (1985) for catalyst design. The application was limited since the development and maintenance of the system was expensive.

In the 90s the Era of Neural Networks (Venkatasubramani) started representing a change in the general approach from top-down to bottom-up. This was enabled by having a complex graph with many layers being used to learn patterns without needing to translate the human constructs into logic. This new approach (Gani) r,esulted in large advances in modeling, system control, error recognition and product design. The progress in development (Venkatasubramani) did not translate into impact in the industry since there was not enough computational power and data, and the problems that could be solved with ML could also be solved with model-predictive control.

In recent times there has been increasing computational power availability, which allowed for the digitalization of society. This has been described by various terms such as Industry 4.0, the Internet of Things or smart industry which proposes a symbiotic relationship between human labor and computers to maximize the work efficiency of the expensive humans by replacing them with computation. This is combined with the globalization which enabled larger populations to take part in the chemical industry as well as generating higher demands. The advances in machine learning have enabled the automation of various tasks such as image recognition (Venkatasubramani), protein prediction (Jianyi Yang) and automated driving (Chan) where it was previously thought that human intuition was needed for the task. These developments give the indication that the new methods like Deep Q-Networks (DQN) could allow for the solving of problems which are impossible with the classical methods. Mixed approaches where MINLP was used as an evaluation system for the DQN (CD Hubbs) have been developed for the scheduling of chemical production. This is quite promising since it allows for the evaluation of solutions compared to a known local ideal without needing an analytical global maximum as will be used in this paper.

Very similar work has already been done (Laurence Midgley) which is being used as a guideline for this project and further improvements will be made on it such as more accurate data and exact simulation of the environment by Aspen.

## 1.3. Research questions

The main research question can be stated as: “What are the possibilities of using RL methodologies to automate the design synthesis of distillation systems using the Aspen environment?” Following from these secondary questions arise such as: ‘’Which parameters will need to be fixed to simplify the problem?”, “Which environment is most compatible to Aspen plus?”, ”How do computational needs scale to the complexity of the problem?” and “Which type of reward should be used?”.

## 1.4. Aims and Objectives

It will be attempted to answer these questions by doing literature research regarding the different techniques which have been developed to solve similar problems and what their limitations are. After acquiring an overview, a case study will be chosen to serve as a testing ground for experimentation and by acquiring the necessary skills in Aspen, Python and VBA the generation of process design will be automated. Aspen acts as the simulator of the process, Python as the data analyser/decider and VBA as the interface between Aspen and Python. For the reinforcement learning software Keras TensorFlow was chosen and it will train an agent to design a process.

There are 3 phases to the modelling process:

1. Demonstrate that Aspen Plus can be automated via Python

This will require crafting an interface between the two programs which allows for rapid changing of the simulation in Aspen Plus. The result of this is the capability to change simulations and retrieving the results from Python. This needs to be as rapid as possible since there will be many thousands of simulations done for each time the agent is trained.

1. Demonstrate proof of concept that RL can be used in principle for the design of distillation sequences

Since it has been previously proven (Laurence Midgley) that such a synthesis problem can be solved these results need to be replicated so that the parameters can be changed to optimize the learning. This will also require a complete redesign of the previous code.

1. Show the potential and the limitations of RL and give recommendations for further studies

After the capabilities of using RL and Aspen are combined in Python the difficult work of figuring out what parameters are useful for training the agent and optimizing the learning environment starts. In this final step the research question will be answered and the limitations of the program will be found.

## 1.5. Reinforcement learning (RL)

### 1.5.1. History of RL

Reinforcement learning has been used for a wide range of applications with booming success and allowed for the completion of seemingly impossible tasks for computers. An example for this is the defeat of Garry Kasparov in 1997, a reigning world champion in chess. The game of chess was thought to need human intuition for playing and thereby be unreachable for computers. This process of pushing the boundary of what computers can do has continued with self-driving cars becoming a reality in recent years. Even the field of protein folding which has been puzzling computational biologist for the last half of a century has been making use of these methodologies in AlphaFold (Service) reaching a median score of 92.4/100 in the CASP competition.

### 1.5.2. Markov process

The mathematical foundations for the models, which will be applied in the creation of this paper, are based on finite Markov Decision Processes (MDP) (Richard S. Sutton). These MDPs are characterized in having a few key properties, which allow them to be used for solving a wide variety of problems.

1. Sequential choices
2. Delayed reward
3. Case dependent actions
4. Predictive capabilities of future rewards

A general formulation for what defines a Markov process is, that the probability of the next state exclusively depends on the previous state and that there is no path dependence on how the point was reached. A process is found to be Markovian (Richard S. Sutton), if the following is true:

𝑃[𝑆𝑡 + 1|𝑆𝑡] = 𝑃[𝑆𝑡 + 1|𝑆1, … , 𝑆𝑡] [Equation 1]

With St being the state at current t, in the deterministic perspective of the world, this is a self-evident truth. Whether absolute determinism is the right way to describe the world or not, it does seem to be applicable for a wide range of problems. While considering that no complete knowledge can ever be reached there are still some cases, where state predictions are sufficient to capture reality in a simplified form. If this is the case a stochastic control process can be defined with the following 4 variables (S, A, T, R):

S is the state space which encapsulates all properties (s) of the world which will be measured, and the action space (A) will have a list of all possible actions (a) which can be selected. The state transition function 𝑇 = f(s𝑡 , 𝑎𝑡 , 𝑠𝑡+1 ) gives a probability for any state, action and future state combination. This is thereby the internal representation of how the world reacts to each action in each situation. R is the reward function, which can define a continuous set of values for each transition which represents our personal preference between one state or another. These values are commonly bounded between an min/max out of reason which will later be elaborated on.

### 1.5.3. Components of RL

The method, which RL uses to learn, is very similar to that of us humans - and with that in mind, the metaphor of raising a baby will be continuously used to explain the properties and behaviours of this system and how we can shape it to do our bidding.

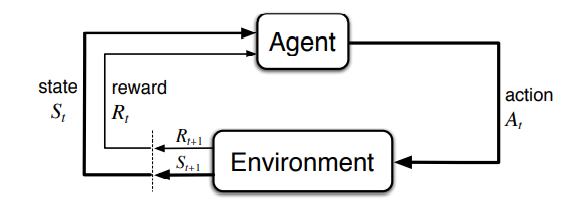
There are 3 main components in a RL system and their metaphorical equivalents: the agent, which in this case is the “baby”, the environment, which is the “world” and then the “parent” that takes the place of a reward function. In most cases the environment and the parent are one and the same thing, but here they will be separated for ease of explanation.

Figure 2 RL learning process (Laurence Midgley)

#### 1.5.3.1. Agent

In sequential decision (Richard S. Sutton), making the agent has the goal of making decisions in a clearly defined environment, where S, A, T and R are computable and by doing so maximize a cumulative reward. Since the child has no idea about anything in the beginning, it will have to learn the appropriate behaviour by using trial and error repeatedly and noting its reward. This cycle of getting a state, making an action, receiving a reward and then repeating it is shown in Figure 2. Over time the child/agent will learn, which actions will result in which rewards for each state and this knowledge is called its “Policy”.

Policy: [Equation 2]

if Policy were perfectly trained: [Equation 3]

This policy in contrast to dynamic programming has no knowledge of the actual environment. The child does not know how or why large heat transfer into skin can damage its body, but rather it just learns a simple policy of: “IF in front of the fire: touch hot stuff = 10\*bad” or “IF in front of food: touch hot stuff = 2\*good”. These policies will later be “remembered” in various neural networks and as the agent learns more, its policy will become more sophisticated. This starting from nothing-approach is also one of the reasons, which allow the RL methodology to be applied to such a versatile range of tasks with only minimal human input into the design. In this case the agent will not know anything about volatility, mass balances or any other human idea of distillation columns and it will derive a predictive model about column sequences exclusively via empiricism about the data it received.

#### 1.5.3.2. Environment

The environment is fundamentally a function or program, which computes the physical properties of some interactive system, which has an initial state (S0) and which can take an action input to find the corresponding new state, which resulted out of this action. In most cases, it is based on a physical model of the reality such as energy balances, but it can also be completely synthetic such as the Super Mario world. This environment could also be the physics of the real world for example in a pilot plant. Temperature and pressure changes could be actions by the agent and these would result in changes in some measurement of the pilot plant which might be used to calculate the hourly production. This could then be used as the reward function. This type of environment has the advantage of being extremely precise since all factors which might influence the results would be included - but the large drawback is the inertia of the system which would result in sluggish learning as well as high costs.

#### 1.5.3.3. Reward function

The “parent” metaphor is not necessary for many states, since “pain” and “happiness” are preprogramed indicators for good/bad in humans. This is different for more complex problems such as “Why should I do homework?”, where the reason only becomes apparent many years later. To prevent this the “parent”, aka reward function, will be assumed to be a perfect representation of good and bad, which knows exactly, how much pain is caused by doing homework vs not doing it. This also makes it clear, why the reward function is the most crucial part of the training, since wrong incentives could have devastating results. In this case an economic evaluation of investment, utilities and other costs will be used as a reward and in the section about reward function in Chapter 2.4 further elaboration will be given on how it specifically works.

With the sequential decision making it is crucial to not just maximize immediate reward, but also to take deferred gratification into account. Just like in a human there needs to be a choice made for how these two types of reward are valued compared to each other. This will be done by using the discount factor 𝛾. The child would like to play outside and get the quick and easy reward, but the “parent” reward function has the capability to predict the future. Thus, it has the ability to take the improvement in life quality, which hopefully resulted from doing homework, into account. The concept of deferred gratification is initially unknown to the child and only with time and some experience it has comprehended enough about the world so that the child’s and parent’s expectations align (aka training is completed).

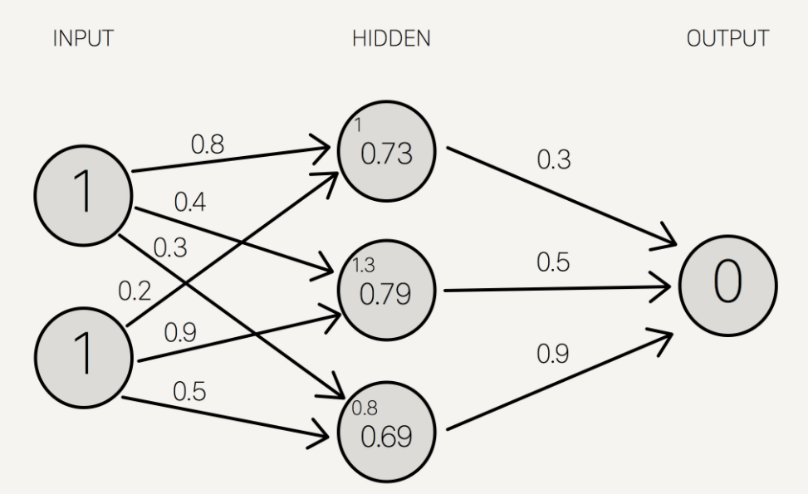
With this new variable the reward function is adapted to the Bellman equation for Q:

Q(𝑠, 𝑎) = 𝑟(𝑠, 𝑎) + 𝛾 max 𝑄 (𝑠 ′ , 𝑎) [Equation 4]

The Q thereby represents a more complete picture of the reward by including the future. Since only the reward r(s,a) for the current state and action are returned to the agent by the parent, the policy needs to be used as an approximation of the real reward function. This approximation will allow for the prediction of the future rewards. If there are multiple action steps in the future this can be done by choosing the action with the maximum predicted reward of the next state.

With enough training the policy should be able to predict the rewards for each action-state pair and thereby be able to choose the action which has maximized rewards. This is called the greedy method (Richard S. Sutton) , since the agent would not choose to deviate from the pathway of local minima that it has discovered and always greedily takes the small but reliable reward it receives. If this greedy method would be chosen every single time, it would become impossible to learn, since there would never be any new experience. “Exploration” is the term used to describe the agent when he just choses some action at random to find out what happens. This is implemented by using the “learning rate” epsilon variable, which describes the probability of taking a random action and thereby exploit and explore simultaneously. In the baby metaphor it has the capability to make sounds and when it cries, people give it attention and try things to make it stop but by random alterations of its babbling, it is able find some combination of throat muscle action like saying “hungry” with much more reliably results in food. If the learning rate is too low then there will never be any noticeable change, whereas if it’s too high then none of the previously learned words could be used, since they would be drowned in random permutations and not be able to find a strong correlation between the two factors.

### 1.6. Deep Q network (DQN)

The policy, which predicts the reward for each action in every state, would become exceedingly large, if the estimated reward would be simply saved for each state and action which was tested. If this is possible, it is called a Q-table. If the state/action space are too large or even continuous this is no longer possible, since the table would become incomputable. By utilizing a neural network (NN) it is possible to select the level of complexity, which the approximation function can have.

This will be the “brain” of the agent. A neural network is based on models, which have been developed for modelling the behaviour of animalia neurons. This has been of limited success (Nikolaus Kriegeskorte) but the mathematics, which were developed for this, turned out to be quite useful. The basic building blocks of a NN are: geometry, layers, nodes and parameters.

Figure 3 Example for DQN with nodal activation and weight factors (Miller)

* **Geometry:** (Input, Output + Hidden Layers)

A neural network is generally structured in layers which each have many nodes inside of it. These layers are structured vertically in Figure 3. Each node has a certain activation which simply refers to the current value of node which is written in the circle in Figure 3. There is also a weight factor which connects each node to every node in the following layer and they are visible on the arrows connecting the nodes. The precise method for calculating the activation will be given in 1.6.1. These layers are subdivided into the input, output and hidden layers. In the input all variables of the state are fed as a number, which in the example above would be 1 for both inputs. The output layer has the size of the action space and its numerical value is equivalent to the expected reward of this action for the current state. The hidden layers are there to add more parameters, which can be fine-tuned so that the outputs are comparable to reality.

* **Layers:**

The layers can also be systemized more generally into convolution layers, deconvolution and fully connected layers. The first and second are used to change the number of nodes between the different layers, which is useful, if the action or state space is too small or too large. Convolution is for example used, when training agents to play computer games, since the brightness and colour of each pixel represents a state (1024 x 768) and there are only a handful of actions (← ↓ ↑ →). Here the convolution layer takes ~780000 nodes and compacts it 4.

The Deconvolution layer does the reverse and an example can be seen above, where the 2 inputs are transformed into 3 nodes in the hidden layer. This can be used to get more complex models, if the state information is sparce. The last type are the fully connected layers, which have every single node connected to every single node of the following layer, which is sometimes also called a “dense layer”.

* **Biological Metaphor:**

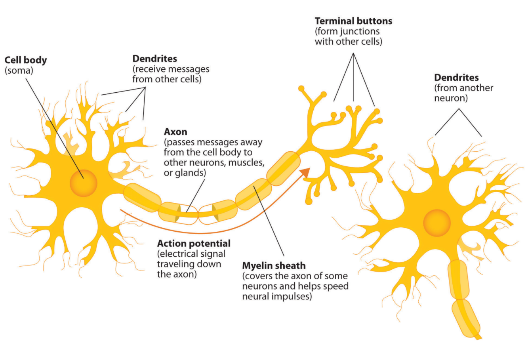
 In the following paragraph the calculation for the activation will be shown. Since the calculation is not very informative the biological metaphor will be expounded upon to enable better comprehension. It should be noted though, that this explanation does not stand up to the state of art of neurological science and it should only be used as a scaffolding. The entire NN is supposed to represent a part of neurological computation infrastructure of living things.

Figure 4 Diagram of biological neuron (Stangor and Walinga)

The NN is composed of a sensory input, which measures something and sends an electrochemical signal to the neuron (aka a Node), where the intensity of the activation represents the size of the variable in question. Each sensor is detecting one variable about the environment and gives its neuron the activation value. Here the idea of the input layer starts coming in, since there must be some interface between the sensors and the computation. This signal is transported through the neuron until it reaches its axon terminal, where it connects to other neurons. The high number of different connection points from each neuron is represented by the dense layers, where all the nodes are connected to each other. In the transmission from one node to another the weight factor between this neuron and every other one gives the strength of amplification of the signal. This idea of weight factors is based on the Hebbian theory that training neural pathways results in higher efficiency and thereby in higher weight factors. In comparison to this, the bias is a minimum excitation, which is needed for transmission, and it can be seen as an activation energy or as a transportation resistance within the cell. With the neuron having many inputs which are concentrated into a single chemical signal within the cell and then propagated to the other neurons it becomes clear, why the interconnected nodal geometry of the NN was chosen in that way.

#### 1.6.1. Calculation of Nodal activation:

First the notation of each node and weight factor will be defined:

Node **activation:** **weight factors:** **Bias:**

With this nomenclature determined it is possible to generate a formula for the nodal activation.

[Equation 5]

For the zeroth node in the first hidden layer from the figure above this gives:

from Figure 3:

The normalization function is used here to prevent the activation from becoming excessively large. Since animalia neurons have a semi binary on/off state it was thought that the sigmoid function would be a good normalization function, but in the actual DQN a rectified linear unit will be used, since it has a faster convergence. This normalization function is also commonly called activation function since it determines which neurons should activate and which should not.

After doing this calculation for every single node in the first hidden layer this process is repeated for each consecutive layer, until the activation value of the output nodes is found. Since this iterative description is very cumbersome a Matrix form is made:

[Equation 6]

This simplifies to: [Equation 7]

With this formula it is now possible to calculate the activation of each node for any given input as long as the weight factors and biases are given.

#### Training the NN:

When first using the agent all the weight factors and biases are initialized at a random value which is most likely not very good at doing predictions. After starting the training cycle shown in Figure 2, the agent gets the initial state which in the case example is zero in all states, since there are no streams yet. Using equation 7 the action with maximum predicted reward is chosen as the action (unless it is exploring, because then it is random) and the simulation is run. Aspen will find out what the new state is, the reward function will determine the reward, and this is compared to what the agent thought the reward would be.

[Equation 8]

This accuracy function will give an indication for how well the NN was able to predict the reward of this action state pair. This accuracy function is usually called a “cost function” in literature. This accuracy is remembered for many runs and once enough data for a minibatch is collected a process called “backpropagation” is used to find the gradient of the accuracy function relative to each weight factor and bias. This is useful since it shows the most efficient method for changing each parameter to minimize the accuracy function.

[Equation 9]

With the old weight factors altercated following the equation below it should improve the ability of the agent to predict the rewards.

[Equation 10]

This does not work for more complex networks since changing the weight factors exclusively in the previous layer does not teach the agent, if there was a mistake in the first layer. The exact algebra for calculating the optimal change in the parameters is outside of the scope of this paper. It is advised to watch: the Neural Network series by (Sanderson) for an explanation of this.

# Literature research

## 2.1. Case study

In the current case study it is the goal to separate a given feed with known composition with the help of a sequence of distillation columns. The values of the case are based on previous work by Laurance Midgley and should be adaptable to problems with differing values - but proving this will be outside of the scope of this paper. The formal problem definition by Midgley was:

“Given the hydrocarbon feed, as defined in Table 1 below, select a sequence of DCs (for each of which a light key and a light key split factor are specified) in order to optimally satisfy a pre-specified performance criterion.”

The simulation of the process will be done using DSTWU in Aspen Plus to predict the physics inside of the distillation column since the case fulfils the assumptions listed in Chapter 4.2.

Table 1: Composition of Feed for Case study

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Compound name** | Ethan | Propene | Propane | Butene | N-Butane | N-Pentane |
| **Feed flow [kmol/h]** | 3 | 5 | 12 | 2 | 9 | 7 |

## 2.2. Search Space

In the following chapter the complexity of the sequencing problem will be expounded upon and what implications this would have on the runtime. Following this, various methods for increasing the efficiency of the search with the help of algorithms are shown.

The space of possible action combination is the mathematic set of combinations which might be chosen for the sequencing of the columns. The size of this set is called Permutations and it is described by the following function:

[Equation. 11]

In the tree representation, which can be seen below in Figure 5, the number of splits after each node is the number inside of the brackets in the formula above. In the case example that value would be 60 since there are 6 compounds with 10 outlet streams which yields around 46.6 billion combinations.

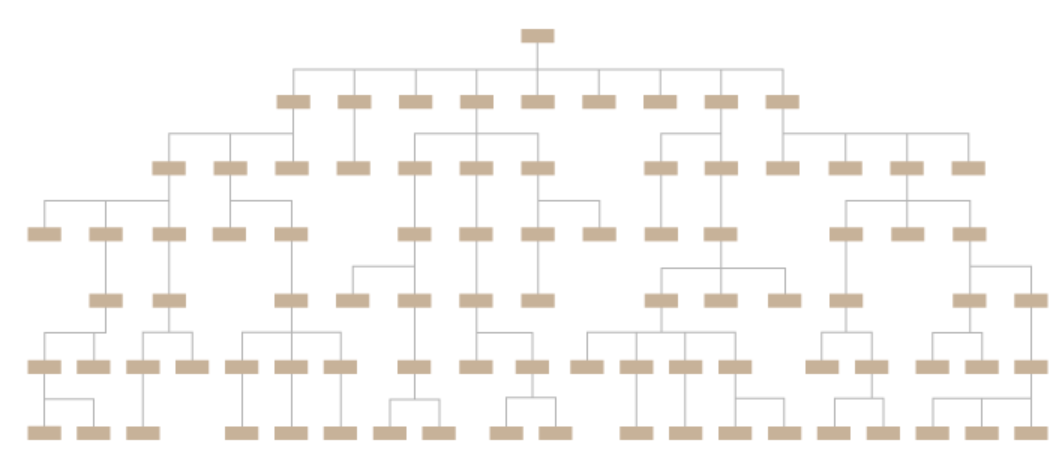


Figure 5 Tree representation of Permutations (Ribecca)

### 2.2.1. Adding more parameters

Just deciding on the light key and the geometry of the sequence is usually not the only step in designing a separation process. So how would the number of permutations change if some more details would be added to the system? One single added parameter such as the type of trays with 3 different types would result in around 3.4e+13 permutations which is around 730 times larger than the one without the tray choices. This nonlinear relationship between the parameters which are chosen and the number of possibilities can be seen in the Equation 11 above and it gets worse the more columns are being used. The table below shows some rough estimates for how large these numbers can get when 6 columns with 6 compounds are used with more parameters.

Table 1: Permutation and Runtime for different number of parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Extra parameter | Number of choices in parameter | Possible Column designs | Permutations | Aspen calculation time in years |
| Compound + Streams | 6 and 10 | 60 | 4.66 e+10 | 739,7 |
| Tray types | 3 | 180 | 3.40 e+13 | 539 thousand |
| Packing | 3 | 540 | 2.48 e+16 | 393 million |
| Construction material | 2 | 1080 | 1.59 e+18 | 25 billion |

### 2.2.2. Runtime length

While running a single Aspen program an average Aspen simulation takes around 0,5 seconds to calculate the simulation. This means doing the simulation for every possibility would take around 23.3 billion seconds which is equivalent to around 729,3 years. This is obviously not the limit since there are methods for increasing the computation such as multithreading where multiple Aspen programs run simultaneously, and such improvements will be expounded upon in the discussion. As it can be seen above the number of possibilities tend to get out of hand rather quickly which means that just increasing the computational power will not suffice for finding the solution and other methods will need to be found to explore this space.

## 2.3. Algorithms for Space search

Since this is a common problem in computer science, solutions have been found to reduce the space of possibilities with the usage of certain rules and algorithms. The following algorithms do this in different manners and their approach will be shown in the next few paragraphs:

### 2.3.1. Heuristics and Rule based limitations:

The most common method which a chemical engineer would use is logic and experience to limit this space of possibilities. These are over time encoded into heuristics which have a broad usability. The first limitation would be that the actions are limited such that only ones which are physically possible are considered. This would limit the number of possible feed combinations to 16. By using further rules such as only being able to pick every lightkey once, the space of lightkeys would be limited down to 5!. Another rule which could be applied would be that the lightkey can not be chosen when there are only trace amount of it in the feed. The last two proposed rules would generally be useful but they also limit the results since there could be cases where there is a more ideal solution prevented by these rules. In the literature Thompson & King developed an equation which determines possible lightkey and feed choices in dependance of the number of compounds. They have also designed a classical logic and heuristic based system for finding possible sequences which are promising.

Possible sequences = [Equation 10]

### 2.3.2. Branch and Bound

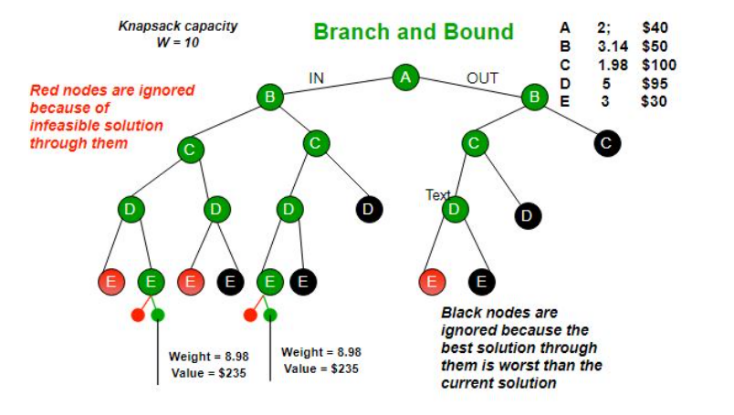
Instead of going through the entire tree from one side to another it is possible to ignore certain branches which seem unlikely to be successful. This is determined by estimating an upper and lower bound of the specific branch which can be found by going through the tree in a limited number of pathways, calculating their fitness and assuming that these would be representative for the side arms. By doing so the number of branches can be radically reduced although this does not guarantee that the optimal solution will be found since it could be possible that the rewards are non-continuous.

Figure 6 Structure of Brach and Bound on a Tree diagram (GeeksforGeeks)

### 2.3.3. Gradient accent

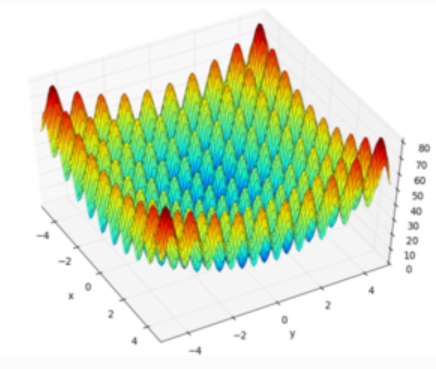
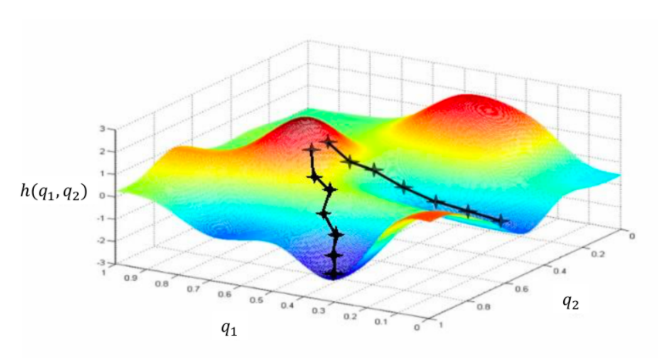
For each possibility in the two dimensions (lightkey, streamnumber) there is a certain reward value and it is the goal to find the possibilities, which maximize this value. When plotting some imaginary data in a 3D graph such as below in Figure 7, it becomes apparent, that there are 2 locations of maximized reward. As this graph resembles a mountain range, where the goal is to reach the peak, it becomes apparent, why the methods is called: ‘Hill climbing’. In hill climbing an initial random choice is made, which places a point somewhere in this map. By calculating the reward in the surrounding spaces, it is possible to go to the neighboring location with the higher reward. This should mean, that the point took a step up the hill. If this process of evaluating the surrounding space and then moving up the hill is repeated, this will result in getting closer and closer to the maximum. As it is visible in Figure 7 below, this entire process needs to be repeated for multiple runs to make sure, that all the different maximums are found. This can be rather difficult, if the space has many local minimas such as the Rastragine function in Figure 8. This approach only works with continuously differentiable functions which is not the case for the given space of possibilities which means that it can not be used here.

Figure 7 Gradient ascent in 3D (Hylton) Figure 8 Rastragine function with many local maxima (Pagmo)

### 2.3.4. Evolutionary strategy

The field of evolutionary algorithm (Xinjie Yu) is based on the idea, that competition and reproductive pressures cause evolution in a population. Natures marvelous complexity is an example of this process and the capability to adapt to almost any environment is a model for this idea.

The first generation consists of a large set of randomly composed individuals, which span the entire space, which is supposed to be searched. A subset of these will be evaluated via a “fitness function”, which determines, how well each individual performed in the task at hand. After the evaluation of these individuals some genetic operation is performed, which changes the internal parameters of the subset. The three most common ones are: mutation, cross breeding and diversity maintenance. The mutation just changes some parameter randomly and thereby counteracts the convergence onto local maxima and thereby aids in exploring the space of possibilities. Cross breeding is mimicking the mitoses of living cells, where the chromosomes of the parents are spliced together. The individuals with the highest amount of fitness have better than random information about the environment and by mixing them, their data is combined in the hopes of generating one, which can combine the understanding of both parents. This type of elitist breeding pattern, where only the most effective ones will procreate, tends to result in genetic homogeneity by converging on a local maximum. Since this prevents any further solution exploration, it is crucial to maintain some genetic diversity within the system.

## 2.4. Machine learning

Machine learning (ML) (Richard S. Sutton) has the goal to convert complex real-world problems, which have an indefinite number of parameters and interactions, into some mathematical representation, which should resemble the real-world phenomenon and is simple to compute. This conversion of systems enables it to predict problems without any understanding of the underlying physics or even what is being regarded. This enables the recycling of methodologies for seemingly completely different problems, such as playing Super Mario and Tetris. This recycling of methodologies is attractive, since it would minimize the labor needed for adapting it to alternative separation processes such as absorption columns. The promising development in this field gives hope, that it might one day be possible to do complete design of a much more complex problem in the field of chemical engineering.

The fast-changing field of ML can be subdivided into 3 categories: supervised, unsupervised and reinforcement learning, which will each be presented.

### 2.4.1. Supervised learning

This is a type of ML, where a known data set of solutions as well as data are paired (Richard S. Sutton). An example for this would be a catalogue of pictures with either cats or dogs, with the labels containing the solution whether it is a cat or dog. This is called a training set. While training the machine only receives the brightness/colour of each pixel in one of the picture without knowing the label which contains the result. By using its internal function, these brightness values are converted to a numerical output of 0 to 1, which gives the confidence of the picture showing a dog or cat.

After making this prediction, the result is compared to the label and the parameters in the function are changed to give a better result. By going through the different examples of inputs and outputs and iteratively changing the function, the function will get better at doing predictions. This has been found to be very successful at identifying underlying properties as long as the data set is large enough. These methodologies cannot be applied in the case example, since there is no readily available set of distillation sequences, on which the agent could be trained, which means, that the simulation would have to be generated which is very cumbersome for this case.

### 2.4.2. Unsupervised learning

In supervised learning (Richard S. Sutton) there is always a solution available for each input, which means, that if the internal representation is unsure about the solution, it could always ask for the true label. This is not the case for unsupervised learning, where the agent is initially trained on a data set, which is labelled. But as soon as the agent is getting better, it will use its own predictions to generate labels and use those to enlarge the dataset, on which it can be trained. This is commonly used when the size of the data is not sufficient for the training. Since there is less control about how good the predictions are, it is possible, that the predictions after training do not function, once novel data is introduced. These two types of learning are also used in humans, where the supervised learning resembles doing exercises in the classroom, where each step is explained and the solutions are known. Unsupervised learning resembles homework, where the personal confidence of one’s solution determines, whether one will ask the teacher about it the next day or not.

### 2.4.3. Reinforced learning

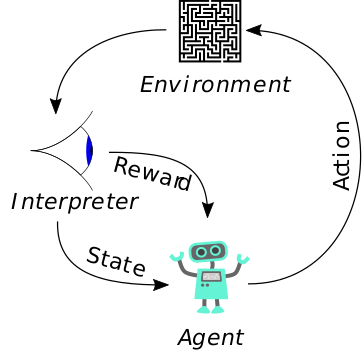
Reinforcement learning is a methodology, which is used to find out, what kind of action in a predefined environment would result in a maximized reward. Compared to the two previous methods, there is no data set necessary for the training and the data is generated in the process of training. This technique is useful as long as the results of every action can easily be calculated or measured. After training the ideal theoretical action should also be ideal in the real world, if the model underlying the environment represents reality. This means, that after the agent is fully trained, he has an understanding of what would be the ideal action for each state to get the highest reward.

Figure 9 Circular learning for RL (Laurence Midgley)

The picture above describes the process of learning. Initially the agent starts out with a random internal function, which yields an action, which will influence the environment in Aspen.

Aspen will create a model for what kind of influence this has on the design and return the reaction of the environment. The interpreter or the reward function will determine how good or bad the change in state is and it returns it together with the new state to the agent. The agent now knows how good the previous action in that state was, and it will do more actions to change it - continuously trying to learn, what kind of actions are useful for each state and which ones are not.

By having the agent make choices which influence the state, this could be imagined as the agent choosing which dataset it wants to enhance its understanding. This increases the efficiency of training substantially, since the agent partially selects the data, it would need to learn most effectively. Since the iterative returning of states is well fitted for the consecutive designing of distillation columns, this is the methodology, which will be employed to solve the sequencing problem.

## Reward function

The reward function is one of the most important parts of the RL system since it determines what the agent defines as good or bad. Without any human intuition the agent is willing to try anything which is within its action space to get a good reward.

In this case the reward will be modelled on the earnings per hour which are given by the following equation:

[Equation 12]

This is closely related to the typical function for maximization which is used in chemical engineering (Chaves) with the novel term of punishment being given, if the agent exhibits unwanted behaviour such as generating an error or choosing physically impossible actions. Just like when raising a child, it might not be a good idea to give an overly excessive punishment when it came home late after playing soccer, because it might result in a misunderstanding, where the child now thinks that it is getting punished for playing soccer. This “trauma” would then result in it never playing soccer again even though this was not the reason for punishing it. In order to prevent wrong conclusions, the weight factors can be changed, so that different errors are punished differently and therefore the resulting reluctance of choosing a similar action can be scaled accordingly.

### Income

The income is calculated by finding the quantity and value of each of the outlet streams. The quantity is easily found since Aspen is doing the simulation, but there is little data available which describes the value of mixtures, since the products are usually sold at high purity. To circumvent this, two equations are presented with the first one only giving the stream a monetary value, if it has a molar fraction of some value, the other one giving it depending on the square of concentration. This means that purity is valued very highly but even below the purity standard some value is maintained by the mixture.

Version 1:

Version 2:

[Equation 13]

The molfraction and molflow are listed in a 2-dimensional Matrix with stream numbers and compound names being the two axis which means that this determination of the value concentration needs to be applied for each point in the Matrix.

The overall equation for the income is the sum of each compound in each stream as can be seen below.

[Equation 14]

### Expense

The Expense can be defined as the combination of the fixed costs and operational costs.

[Equation 15]

Which can be equivalently formulated as:

[Equation 16]

Cost of utilities is found by using the duties which the distillation columns need and calculating the heating value and the costs connected to that. The equipment costs are found by calculating the sizing of the equipment and then using these values to estimate the capital costs. The costs for the raw material are very hard to estimate and thereby they are initially simply chosen to be very low.

### Total Annual Cost (TAC)

In literature there are various ways of estimating the annual cost for the operation of a distillation column, in this case one from (Errico) will be used. Here the number of stages(N), reflux ratio (R), plate efficiency(E), working time(Y), vapour velocity (G), vapour handling capacity (G’), annual incremental investment for condensers (C’) and cost of steam/coolant (C’’) are used. These are all combined in the following manner to yield the TAC:

[Equation 17]

The value of the constants can be found in the appendix.

### Punishments

There are multiple types of punishments which need to be given to the agent. The punishment in general has the purpose to give the agent some type of external change in what is rewarded and thereby change the incentive which it has. This enables the agent to learn what would generate errors, where he oversteps boundaries and just generally to influence the behaviour.

Since different error types are of varying negativity, they each receive a weight factor which scales the severity of the punishment.

[Equation 18]

In the following list the different types of errors will be explained:

1. Choosing an action which is not available.

Because of the geometry of the NN it is only possible to have a one-dimensional array of possible actions but since the choice of light key as well as stream needs to be made, all possible action choices are:

[Equation 19]

The number of streams change as the agent completes more and more actions but the action space must remain constant, since it is defined by the NNs output layer. This means that the agent chooses one of the 6\*12 = 72 actions, but most of them are not available. An example for this would be: after completing the first column he chooses the bottom of the third column as the feed for the second. This cannot work since the third one does not exist yet. Keras is the current program which is being used for the training of the agent and it is difficult to prevent the selection of such an action within it. If a self-written training program would be used for the selection of the action this could be implemented but by using a library from someone else, this is impossible. Without a punishment around 97% of the actions are not available.

1. Simulation error

The choice in light key may result in an error in the actual simulation of the column and thereby prevent the accurate simulation of the process. Examples for these are: negative number of stages needed for separation, separating a compound as light key which no longer is in the mixture and needing a bypass. Since these errors usually do not prevent the generating of a flowsheet, this is a type of error which can easily be ignored without breaking the program, but they still need to be prevented.

1. Block properties error

This type of error is very similar to the previous one, just that these are not corresponding to running the simulation but rather in calculating properties of the column. Often time the first type of error would result in the second one also having problems, but the separation is necessary, since they can also appear independently. The block properties which are measured in this case are: distillate and bottom temperature, condenser and reboiler duty, number of actual stages as well as the actual reflux ratio. These could be supplementary states which could be fed to the agent, but with simplicity in mind this was forgone. Since some of these values are needed for the calculation of the economics a placeholder is used. If the value cannot be calculated, the number of stages will be assumed to be around 150 and the bottom temperature to be 50 C.

# Methodology

## Notations

Since there are 6 compounds there are 5 columns needed for the separation. This is Nmax and the columns are numbered in the series in which they are created from B1 to B5. Since each column needs 2 streams, each of these streams is called the name of the column and then with T and B for top/bottom. This names all possible streams from B1B to B5T as well as the columns. As previously mentioned in chapter 1.7. the requirements for a DQN is a fixed number of actions and states.

## ASPEN DSTWU environment

Within the Aspen Plus simulation program a choice had to be made which type of distillation program should be used. In this case a short cut called “DSTWU” is selected, where the Winn-Underwood-Gilliland method is the underlying model to simulate the distillation column. This was decided since it has A) few inputs needed B) tends to converge reliably with minimal errors C) is quick in calculating D) all outputs needed for the cost estimation are given. (Gavin Towler)

1. **Inputs**

The inputs which this model requires are:

* lightkey and heavykey recovery
* condenser and reboiler pressure
* partial or total condensing of tops.

The choice in lightkey and heavykey here represent only one choice since it has been assumed that the heavy key is always the compound which is next most heavy to the light key and thereby:

These are useful assumptions since it prevents the agent from making choices which would break the fundamental mass balance of the process. It also does not allow the agent from choosing one compound for both keys, the light key in the bottom and various other scenarios for which punishments would need to be implemented.

1. **Converge reliably with minimal errors**

The Winn-Underwood-Gilliland method converges reliably which is very important, since any type of error would require the agent to learn why this happened and this would take many tries to find reasonable correlations between the specific choices and the creation of errors. With more tries , meaning more computation, it is necessary to minimize those, since this would increase the time the agent needs to learn.

1. **Quick to calculate**

As previously mentioned it is important to minimize the time needed to get the agent trained. This is of high relevance since running the simulation takes the longest time of all the different steps involved in training and a doubling would result in a doubled total training time.

1. **Outputs are sufficient for making first cost estimation**

The outputs which can be found by using the Winn-Underwood-Gililand method are:

* number of stages or reflux ratio
* condenser and reboiler duty
* top and bottom temperature

In the current case the reflux ratio will be defined as a multiple of the minimum reflux ratio. It would also be possible to define a standardized number of stages, but this might make it necessary to have more than 5 distillation columns to reach complete separation and it would open the possibility of having unrealistic reflux ratios. It was decided to use a reflux ratio of:

The duties are needed to estimate the operational costs since the amount of energy which is consumed in the distillation column should be part of the cost function. Further details can be found in the reward function section in Chapter 2.4.

The temperatures are needed for estimating the operational costs of the distillation column, further calculations can be found in the appendix.

**Assumptions**

The prediction of non-key distribution does not give accurate predictions (Gavin Towler) which means that the amount of trace quantities in the top and bottom should not be trusted. This will have to be considered when determining the value of the individual streams which is discussed in the reward function discussion. Another case where the DSTWU column should not be used is, when there is a large nonideality in the liquid or gas phase. This results from the initial assumption that the relative volatility of the feed is constant over all temperatures and pressures inside of the column. The compounds and the compositions are chosen such that these assumptions are true and no azeotropes exist.

## Pseudo code

In the following chapter the structure of the code will be shown which makes the whole theory much easier to understand since most of the problems can be hidden behind functions. In this case Keras will be used for all the DQN training and Aspen for doing the simulation. With Aspen Plus having documentation for communicating with excel Virtual Basic Application (VBA) this program was used as the interface between Python and Aspen. This means that all the interactions between the two softwares need to have a vba function executing it which makes the system far more complicated.

Class: Environment ()

Function: **InputChoice()**

takes the 0-72 index and converts it to light, heavy key and feedstream name

Function: **makeProcessandRun()**

takes lightkey, heavykey, feedname, adds a column in aspen and then runs simulation

Function: **GetOutputs()**

determins the flowsheet and Block properties from simulation

Function: **CheckIfValidaction()**

checks there are any errors or problems with the simulation

Function: **GetReward()**

calculates the economic analysis and adds punishment

Function: **findchoices()**

find out which actions are going to be possible for the next column

Function: **rememberAction()**

remembers actions and other variables which would get deleted

Function: **step(action)**

The step function is the most important one and it has 4 properties: It takes an action as an input and returns the new state, reward, whether the sequence is completed and then a message called ‘’info’’ for debugging. All the previous functions are specific to the case example and the DQN training does not know or use them since only calls on the step function.

Inside of the step function all the previous functions are called which means that calling the step function completes everything which the “Environment” is supposed to do for the agent.

Instantiate Class

The previous block of code is a CLASS which means it is a generalized block of code where the specific values can be inserted later. This insertion of values is done here:

**Env = Environment (**Here all the details are inserted like: Initial molar flow, pressures…**)**

Generate model which will be trained.

**Model = Sequential( )** #Since there are sequential choices

Input layer: **Model.add(State size)**

Hidden layer: **Model.add(Dense(number of nodes, normalization function choice))**

Output layer: **Model.add(Dense(Action size))**

Generate an Agent.

**policy = BoltzmannQPolicy()** #type of Q function which is used … (future costs are considered)

**memory = SequentialMemory(limit=50000)** #how many data pairs are remembered

**dqn = DQNAgent(model=model, memory=memory, policy=policy, nb\_actions=actions)**

Training of the Agent

**dqn.compile(Adam(lr=LEARNINGRATE), metrics=['mae'])** #Adam is a optimizer to make it fast

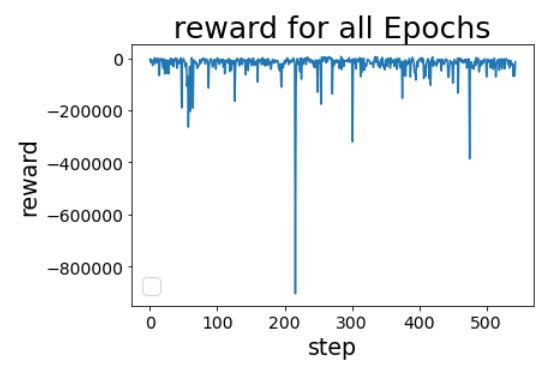
**hist = dqn.fit(Env)** #Train the agent

# Results

There are various methods for measuring how many steps are made. The units from small to large are: action step, valid step, episode, epoch. The action step involves choosing an action and if its legal, this is a valid step. How many action steps are needed depends on what the ratio of legal actions is. Once 6 legal steps are chosen this completes a sequence, which is called an episode - and after around a thousand episodes an epoch is completed. These epochs are used to delete accumulated data as well as averaging the values from the many episodes and thereby giving a more representative picture.

To measure the success of the methodology, the sequences with the highest reward will be compared to the Heuristics, the difference between the methods for evaluating the overall performance as well as the two economic evaluations will be shown. The reward distribution, exploration vs exploitation, punishment effects, runtime analysis and training instability will also be discussed in the following chapters.

## Agent evaluation metrics



When training the agent, the simplest method for evaluating the agent would be to just plot the reward for each episode and then see if there is any improvement visible. This can be seen in Figure 11.

If the reward predictions of the agent correspond to the reward which the environment would give, this would mean, that the Agent has complete understanding of what each action results in. This results in the agent almost always choosing the same action since he is very confident in his model, and he knows that this is the best choice. This process is called convergence and sadly this state has not been reached in the training process but there are still discoveries to be made about what the agent has been able to learn.

Figure 10 Episodic reward for the agent

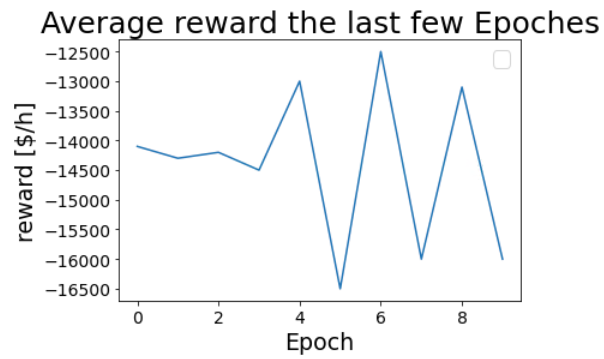
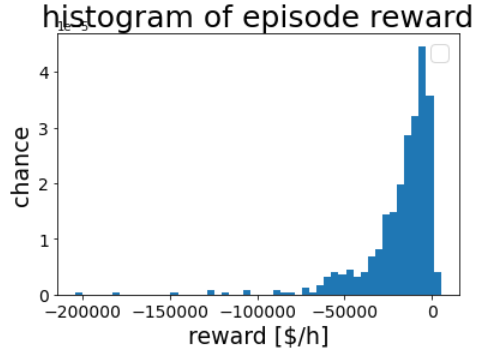
In Figure 10 the reward per episode is plotted and there is a very large range of rewards which span from around 5000 to -800 000. This is the caused by there being much more possible designs which do not make money compared to the ones which do. In Figure 15 is the operational costs visible which is the main reason for the large negative. In this case there is no clear sign that the reward value improved while training. Therefore, an alternative value needs to be used for evaluating the learning process of the agent and one possibility would be the mean or median. The mean is dominated by the rare but very large negative rewards. This can be seen on the left where the low points in the average reward correspond to the negative peaks in the rewards. The median does not face this problem and is thereby much more stable, but it does still face the problem of not representing any of the extreme values. To also represent these parts of the spectrum the maximum and minimum values are recorded for each of the epochs and plotted. Since their value strongly depends on chance they are of little value for the evaluation unless more complete statistical analysis is made.

Figure 11 reward and average reward for 10 000 steps

Figure 12 Histogram of all Epochs

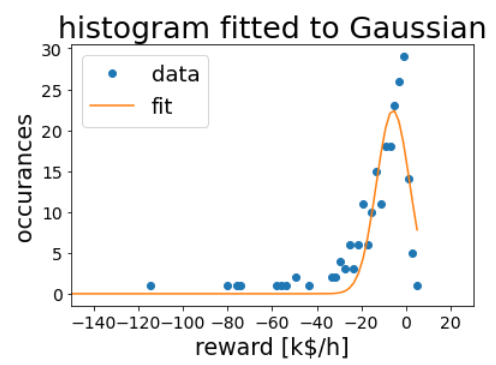
The most complete representation of what rewards occurred would be a histogram since it can give information about the distribution of the rewards across the different epochs. In Figure 12 the histogram of each epoch is combined into one which represents the average distribution across the training. With the goal in mind that the agent can learn about the environment and thereby have non-random actions it might be, that the histogram across the different epochs changes. To evaluate this a Gaussian non-normalized curve was fitted onto the histograms in Figure 13 which used the peak height, peak position and a scaling factor as parameters. The scaling factor would correspond to the standard deviation if the graph were normalized, but here it is just an indication for the thickness of the curve. Since the histogram does not seem to be normally distributed but rather follows a Laplacian or even an extreme value distribution, it would make sense to use these to fit to the data. This has been done but it was found that the automated fitting is much less reliable and thereby it could not be used.

Figure 13 Gaussian fitted to histogram.

## Changes to the model

Following from the initial results alterations to the model need to be made to allow for better learning. These are the changing of the compound value, limiting the minimum reward and making the DQN more complex.

### Change the value of the compounds

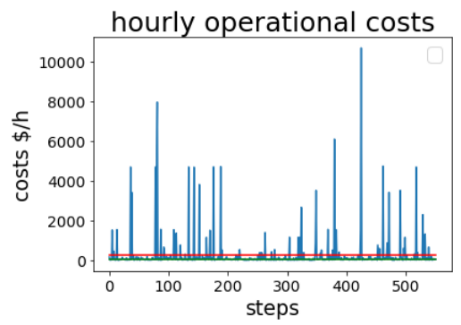
In Figure 15 are the total hourly operational costs shown with the median/mean being plotted in red and green. The operational costs can get very large with the maximum being 10k. These events are very rare and are caused by selecting extremely high reflux ratios of multiple thousand. The cost of buying feed on the other hand was initially around 0.001. This meant that this factor plays almost no role in the determination of the reward which was changed by increasing it substantially such that it is roughly 0.1 $/h. This was done so that using the feed would result in costs, since the agent otherwise exclusively minimized the costs without even trying to do separation.

Figure 15 Operation costs

### Hard maximum reward to limit the minimal reward

As can be seen above there are some variables in the reward function which can get extremely large and doing so can mean that the agent evaluates the preventing of bad cases much higher than finding good ones. To mitigate this the operational costs and capital costs are limited to a boundary variable and the same is also done for the total reward. When the boundary condition is too low it means, that bad behavior can no longer be distinguished from very bad ones and this prevents learning. The boundary also limits the range of the reward predictions, which results in limiting the weight factors inside of the DQN. In many other applications the reward function is also normalized between zero and 1, but to guarantee equal conditions for each training this was not done. Even with this limit in place there are some cases when the reward can become very negative so that the future prediction part of the Q function becomes excessively pessimistic. This can only be mitigated by utilizing a custom DQN training program.

### Simple DQN

The initial DQN consisted of 1 dense layer and around 60 nodes which turned out to be far too simple and it was impossible to recognize any learning patterns with this system. The layers were increased to 4 dense layers with 128 nodes each. This gives a total of around 67k parameters inside of the DQN, which can be adjusted. When increasing the number of layers and nodes the time needed to readjust the weight gets larger. Adding more nodes costs much less computation then adding further layers, since each additional layer requires a complete recalculation of all the weight factors in the DQN while adjusting the weights. Since there are no clear guidelines for the required number of parameters, it was assumed that 67k would suffice, since there was no difference recognizable.

## Punishing results in adaptation

The two main types of punishment are the one for attempting an illegal action as well as creating an error while simulating in Aspen, which is called block error. Without any learning roughly 97% of all actions would result in illegal actions, but learning how to prevent this error is one of the first things the DQN learn. Within the first few hundred steps in the first epoch this success rate increases by a factor of 10 to around 0.307 which remains stagnant for the rest of the learning process. This is caused by many of the potential illegal actions being very easy to recognize such as choosing the output of a column which does not exist as the feed. Since the numbering of the columns is independent from the actions which are chosen this means that the DQN needs to either reconstruct its previous choices or remember them. Since a Markov process is defined as a process without memory this is a very difficult problem for the DQN. It would be substantially easier if the input into the DQN included the previous sequence of actions or even some information about the current sequence numbering. This is also the reason why a further increase in the punishment above 100$/h has been found to not result in any increase in legal actions.

Figure 16 Fraction of legal actions

Overall, the graph in Figure 16 thereby proves that the agent is learning while it is being trained and that these learned properties are remembered inside of the network.

## Agent makes use of errors to manipulate reward function

The problem of aligning the agent’s goals with the ones of the designer can be difficult and the reward function is the only tool available in RL. Since the agent has no idea about what the designer wants, it will exclusively maximize reward in its environment. Any mistake in the allocation of reward or simulation of the environment can result in dramatical results as shown below:

Because of an error in the code any simulation error in the last column would not be considered for the reward. Therefore, the agent was able to define the cost of this column with zero without getting a punishment. This mistake was recognized, learned, and rapidly exploited which ended in the training converging on this error. In Figure 17 it can be seen how the agent is converging on always choosing a possibility where the last column can not be simulated.

Figure 17 converging on simulation errors

It was initially thought that by scaling the size of the punishment it would be possible to prevent any error which does not seem possible. The minimal error fraction is 0.075 even when the only economically relevant factor was the prevention of the error. Nevertheless, this is a very good result since most action combination result in an error and around 5% of the 7.5% could be potentially caused by the agent choosing a random action. Interestingly the further increasing of the punishment led to an increase of the error rate, since the agent is “traumatized”. With these large punishments it will react with large alterations in its weight factors, thereby preventing any learning because all previous experience is overwritten by these changes.

On the other hand, when the punishment for creating a block error was set too low, this would result in an economical advantage in generating a simulation error. This is caused by the operational/ capital cost calculation relying on the tray number and reflux ratio, which in a simulation error are set equal to zero. Therefore, the punishment should always be at least twice the sum of the mean costs for those two factors. For most cases 1000 $/h was found to be ideal.

## Heuristics vs best agent choice

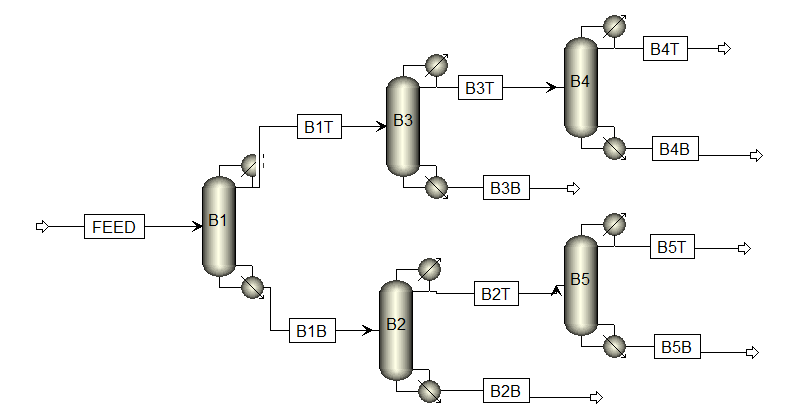
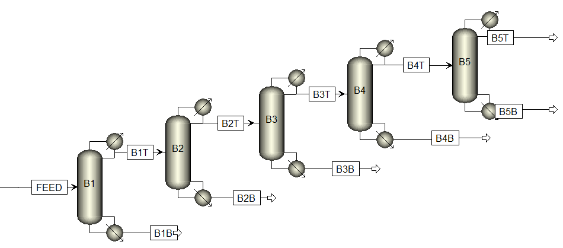
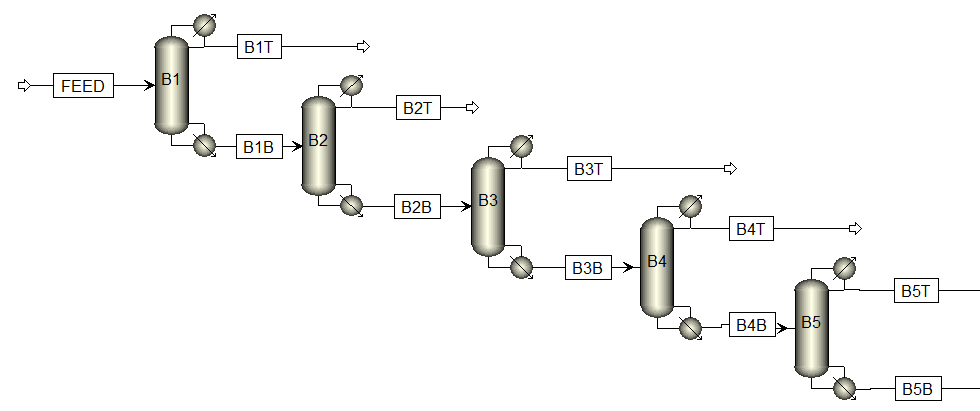
As previously mentioned, it was not possible to have the agent converge on a single ideal strategy which meant, that there was no clear advice which sequence should be used. Instead of waiting to converge at some point, the code can also just remember the sequences with very high rewards and then use those for the comparison with human heuristics. This is necessary to give a rough reference of the quality of the rewards in common industry standard sequences.

There are various types of heuristics which are used to design a distillation sequence. In this case only 4 will be used as a reference for the quality of the agent, and they represent the instinctual knowledge of a human chemical engineer. These do not represent the ideal type of separation and are just comparative cases, but they do have some properties in common:

* Each stream has only one main component
* no errors and no punishment are given
* No ‘punishment for bad action’

### Heuristically solutions to sequencing

The following 4 heuristics (Malone) are used to design a reference for the cases which the agent choses: Separating the most volatile first, balancing top and bottom molflow, separating the least volatile first and then separating the largest component first. These have been modelled in Aspen and evaluated using the reward function.



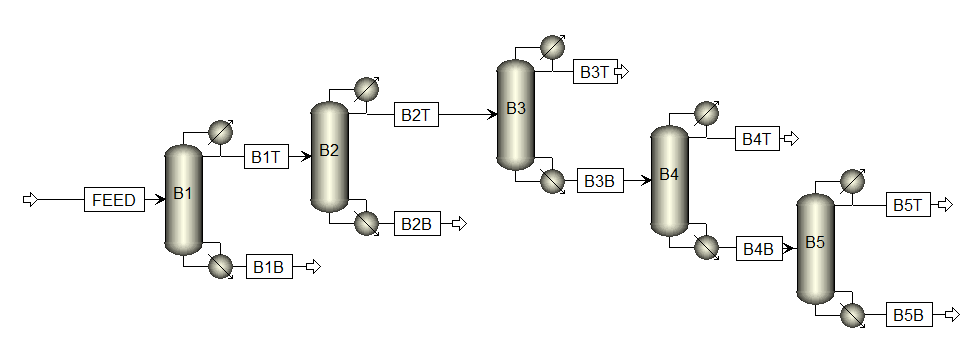
Figure 18 Separating lightest key first Figure 19 Equimolar separation

Figure 20 Separating heavy key first Figure 21 Largest compound first

The Table below shows the output streams for each of these heuristic solutions and the action series can be reconstructed by following the heuristics.

Table 5: Heuristics separations and Rewards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Light key** | **Equimolar** | **Heavy key** | **Largest comp.** |
| **Reward V1 [$/h]** | 2528 | 2365 | 2671 | 2617 |
| **Reward V2 [$/h]** | 352 | 884 | 1487 | 953 |
| **B1 outputs** | ETHAN-01 | Non | N-PEN-01 | N-PEN-01 |
| **B2 outputs** | PROPY-01 | N-PEN-01 | N-BUT-01 | N-BUT-01 |
| **B3 outputs** | PROPA-01 | PROPA-01 | 1-BUT-01 | ETHAN-01 |
| **B4 outputs** | 1-BUT-01 | ETHAN-01 | PROPA-01 | PROPY-01 |
| PROPY-01 |
| **B5 outputs** | N-BUT-01 | N-BUT-01 | PROPY-01 | PROPA-01 |
| N-PEN-01 | 1-BUT-01 | ETHAN-01 | 1-BUT-01 |

### Best Action choices

Table 6: Action choices found by Agent.

|  |  |  |  |
| --- | --- | --- | --- |
| **V1** | **V2** | **Lightkey** | **Feed** |
| 2167 | 1047 | N-BUT-01, 1-BUT-01, ETHAN-01, PROPY-01, 1-BUT-01 | Initial Feed, B1T, B2T, B3T, B4B |
| 2296 | 1443 | 1-BUT-01, PROPY-01, PROPA-01, N-BUT-01, N-BUT-01 | Initial Feed, B1T, B2B, B1B, B4T |
| 2458 | 1410 | PROPY-01, 1-BUT-01, PROPA-01, N-BUT-01, 1-BUT-01 | Initial Feed, B1B, B2T, B2B, B1T |
| 2139 | 959 | PROPA-01, PROPA-01, N-BUT-01, 1-BUT-01, PROPA-01 | Initial Feed, B1T, B1B, B3T, B2T |
| 2379 | 563 | N-BUT-01, PROPA-01, PROPA-01, 1-BUT-01, PROPA-01 | Initial Feed, B1T, B2B, B3B, B3T |
| 2461 | 1401 | PROPY-01, PROPA-01, N-BUT-01, 1-BUT-01, N-BUT-01 | Initial Feed, B1B, B2B, B3T, B1T |

### Comparison between heuristic and agent

The different solutions which have been found should be separated by the reward equation which was used to evaluate them since they do deviate substantially.

In V1 the incentives did not generate high purity outputs and as a result the agent found various combination with purities being around 80-50% while minimizing the operational and capital costs. These had surprisingly high rewards which were on average better then both the heuristics as well as for V2. This does not mean that these solutions would also be applicable in reality since the V1 reward seem less realistic. The V1 also tended to generate more block errors and accept more punishments since its main strategy seemed to be cost minimization instead of sales value maximization.

For V2 the agent would only be able to sell its outputs when it had a minimum purity of 90% meaning that it would have to find a pattern, where a high purity output is generated.

The 4 heuristics belong to a very rare subset which separate all products. Only 32 of the 46 billion combinations can do this and the agent has not been able to discover even a single one of these. This is partially to be expected since the total number of simulations only reached around a million. The human intuition is very quickly able to figure out all the 32 combinations, since it understands from the start, that there are not that many ways to combine columns. This is an unfair comparison though, since the agent does not know that there are columns being combined or really anything about what the actual environment is.

Since the agent has failed to find these complete separations the average, mean and minimum reward from V2 are far below the one for V1 as well as the heuristics. The best cases for V2 were able to separate up to 4 of the 6 compounds which can results in almost comparative sales to V1 since it only separated the most abundant compounds.

## High vs low exploration

When the agent has started to get an understanding of what actions are good or bad it is forced to make a difficult decision: should it choose the best option trying to get the maximum exploitation of the environment, or should it take a random action exploring the space of possibilities and thereby being able to make better predictions in the future?

During every process of learning the exploration and exploitation need to be balanced as was explained in the introduction with the metaphor of the baby babbling. In the RL context does the learning rate variable determine for what percentage of the actions a random one will be chosen and this needs to be determined by the designer.

If it is set too low the process of learning will be sluggish and most of the simulations will result in no new understanding of the environment - but the current knowledge will be repeated and thereby solidified. If the reversed approach is used, so that every choice is random (aka stochastic search of the space) and many things will be tested, no recognizable pattern will be detected. They are hidden like a needle in a haystack. With a methodology using a memory (such as Bounded search) this might not be a large problem, since it can simply save the top 100 solutions and then test those, but this is not possible with DQNs since they do not have memory. The DQN changes its weight factors not with the goal of finding the maximum reward, but to predict all rewards for all actions in all cases. Once it has reached this state is it able to predict the action sequence with maximum reward.

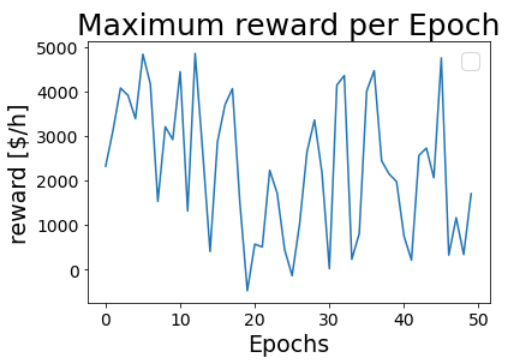
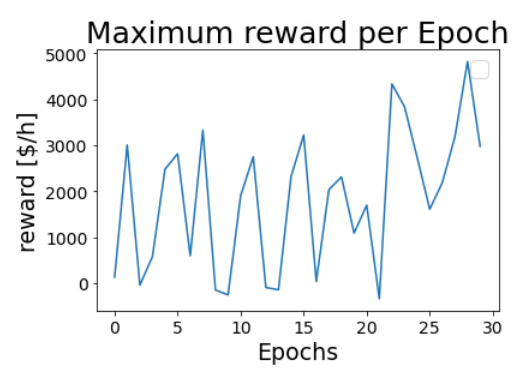
It has been discovered that with exploration below 10% it will converge on a certain branch of the tree and then start to deviate from there and thereby search the side branches from this starting point. The optimum was found to be around 5%, which was then used for most of the experimentation.

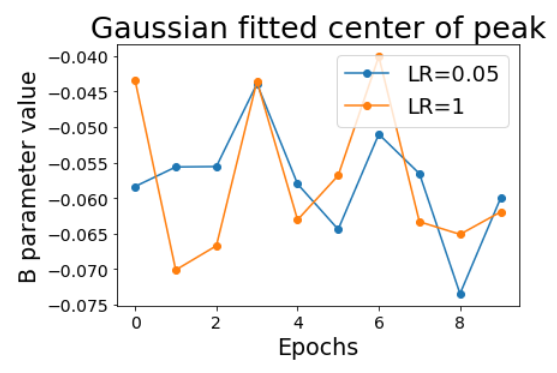
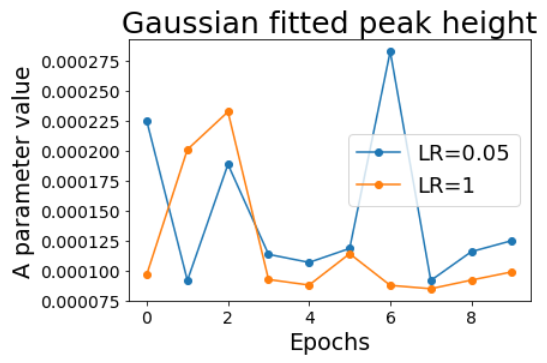
With exploration above 10% there is only noise, since it starts by attempting to predict the entire space of possibilities with only its couple thousand examples.

These values have been found by testing, but it is advised to do some training over a whole range of different learning rates and then find the minimized average loss function per learning rate. There is also the strategy to initially start with a very high exploration rate and as soon as the agent has a rough understanding of the system, it decreases the learning rate, until it reaches zero once it has completed training.

### Comparison to random choices

The large advantage of the RL methodology, compared to the other algorithms presented in chapter 2.2, is that the policy (once it is completely trained and has converged) is able to make predictions about the results of an action series even if it has never seen the results. This makes it very valuable since the DQN prediction calculations are much faster than the ones for Aspen. The reward for the entire space of possibilities can be easily calculated using the matrix equation 7 and the maximum is easily found. Once the agent is trained it can be used to go through a list of possible permutations and see which ones the agent predicts might be viable – and afterwards do the real simulation.

Since there was no convergence or complete training possible the methodology at hand now has the same use as stochastic search which only generates a list of possibilities. To find out if the RL method is more efficient than just randomness, the learning rate was set to 1 and an equal number of training steps was performed to enable comparison.

Figure 22 Max reward for Random actions Figure 23 Max reward with learning

The average max reward for both cases is around 2000, so there is no indication that the maximum reward per epoch is improved by having the agent learn something. The lower median reward might even be an indication that the usage of the policy limited the exhibition of large maximum errors. To find out more about it the histogram will be analysed for both via the gaussian plotting method explained in chapter 5.1. In figure 25 there is no clear trend that indicates that neither the peak height nor its position have shifted with or without learning. This could mean that the agent’s estimations of rewards given by the environment are so far from reality, that they resemble the random distribution. Overall, only very weak conclusions can be drawn from this. The RL approach was not significantly better than just random sampling of the space. This does not mean that the general methodology does not work but rather that the agent was not able to learn enough about the environment to be able to make use of it learned knowledge.

Figure 24 Gaussian fitted to histogram

Figure 25 Histogram comparison for Learning rates

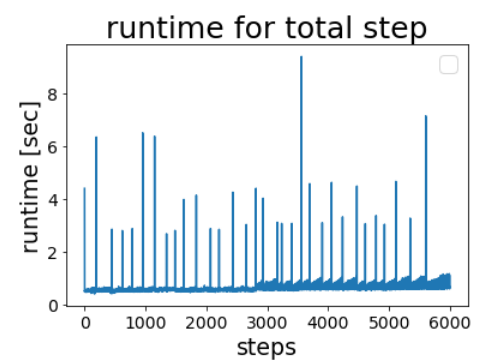
## Runtime analysis

A detailed analysis of how much computation is needed for making a design or simulation is difficult since this will depend on various parameters which are predefined by the hardware being used. The field of computational performance is rather complex and to mitigate this a relative unit will be used for evaluating which one of the steps in the process is currently rate limiting. This will be the runtime which refers to the time the program needs until it ran to completion. This was measured and averaged for various parts of the program and the results can be seen in table 7.

The analysis of the allocation of computational work is of relevance since this allows for the optimization of the process which would result in lower resource utilization and faster learning. Even with the reduced costs of doing calculations in the last decade the exponential nature of complexity requires careful optimization of the program since any inefficiency is scaled up when used on larger problems.

Table 7: runtime averages of different program parts

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Function** | Data input to aspen | Check illegal action | Place column + simulate | Get Aspen results to python | Check for errors | Calc the reward | Find legal actions | Remember actions |
| **Runtime [sec]** | 1e-5 | 3e-6 | 0.42 | 0.28 | 0.012 | 0.004 | 6e-5 | 2.3e-6 |
| **contributing programs** | Python | Python | VBA  Aspen | Python  Aspen | Python  VBA | Python  Aspen | Python | Python |



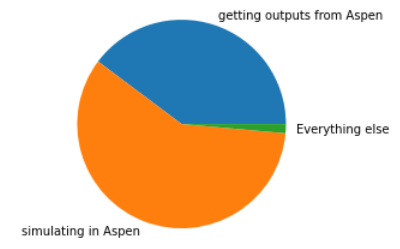
There are clearly two rate limitations in the process of doing a step. The placing of a column and simulation as well as receiving the outputs of this simulation. The first one is mainly caused by the simulation which Aspen needs to do (65%) and the second is the result of the Aspen Python connection being unreliable which results in needing to run another simulation while trying to get outputs (35%). Both steps are limited by one subroutine which is the Aspen.Run() command. The diagram in figure 26 shows the distribution of the total computational time. Besides that, there is an increased time whenever different programs communicate with one another. This is highlighted by the difference in magnitude between the Python and Python + VBA functions. This is most likely the result of the interface created for this purpose and further improvements here might be advisable.

Figure 27 Runtime for retraining the DQN

The overall runtime for readjusting the weight factors is negligible for the entire training time except in regular peaks, where the DQN seem to take a long time. This is caused by the agent not actually readjusting its weight factors after each reward but rather collecting a couple of hundred and then doing the back propagation once. This can be seen in figure 27: the overall average for this is still negligible compared to the other ones, since this process has been optimized by the engineers who developed Keras.

Figure 26 Distribution of computational time

# Conclusions

## Answering Research questions

The first question regarded the possibility of automating Aspen which has been found to be successful. Here the usage of VBA as the intermediator was found to have considerable drawbacks compared to direct Python-Aspen communication. There are still some problems with the stability of Aspen but there have been first solutions found and further ones proposed to resolve it. These can be found in more detail in the Phoenix function. For the question which environment should be used to do the training it has been found that TensorTorch would me more advisable compared to Keras which is currently being used. This is caused by TensorTorch being more easily adapted to use a custom DQN which will be necessary if heuristics are to be implemented to limit the space of possibilities. In chapter 2.4. there were two equations proposed for evaluating the value of the Products. It was found that V1 does not incentivize high purity product but rather just minimizes the costs. Since this is unwanted it is advised to use V2.

There where various parameters which influence the learning and, in the Table, below is the advised value for these.

Table 5: Proposed Variables.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Learning rate | Illegal action punishment | Simulationerror punishment | Max neg. reward | Max TAC |
| Proposed Value | 0.05 | 200 | 1 000 | -15 000 | 2 050 |

## Problems with the methodology

### Runtime

As previously mentioned, it seems that the simulating of the column is one of the limiting steps in the process of learning. The code responsible for these calculations is not accessible since it is hidden inside of Aspen so that optimizing it is difficult. In the current case the Winn-Underwood-Gilliland is the foundation of the simulation, and it has been shown by (Laurence Midgley) that it is possible to do similar calculations much more rapidly which indicates that either Aspen itself or the usage of Aspen was quite inefficient. Unless there can be some method found for improving the simulations, it is doubtful, if the here proposed methodology is scalable to more complex problems. Since Aspen is the dominant software in the industry it might be difficult to switch to other programs such as COCO, unless there is some method for transferring the designs from one system to another.

The time needed per step might be limited for the current setup but there are ways of increasing the steps per time such as having multiple programs run parallel. This would also enable a whole field of other techniques used in RL such as duelling DQN or communicative DQN which have a much higher chance of convergence. It would also be useful if the entire system would be multithreaded: instead of going from A🡪Z step by step, two programs go from A🡪M and M🡪Z respectively. This allows for a doubling in speed because the labour is divided between the two as long as the two pathways are independent and combinable.

### Stability

Stability in the field of computer programs means the ability of the program to execute many times without breaking down or crashing. The problem of stability has been completely neglected until now even though this was the largest problem in the entire thesis and that is because it has nothing to do with RL or even Process technology and yet it is an integral part of the problem. When a computer program encounters a problem, not knowing how to solve it, it will inform the human by putting a message on the screen. When programs interact without a human these pop-up messages are suppressed, and the error message is exported to Python where an analysis can be done for what to do. This is called errorhandling, and it was possible to do this for all the different types of problems which Aspen and VBA faces, except for the critical simulation failure. This would result in the entire Aspen program crashing and not responding until the message was manually clicked. Since Aspen does not allow for the handling of this error it was not possible to just have the system train for multiple days on end but only for around 5-8h on average. This should be considered when designing a program which makes use of Aspen.

Besides this problem the usage of VBA has increased the difficulty of handling the errors by increasing the number of crash locations while debugging. As a result of that it is advised to mitigate this and only use Python and Aspen which would also aid in the runtime.

### Custom DQN program

The current program which was used for training the DQN was Keras which is an excellent program, but which does not allow for altering its internal code. Therefore, it was not possible to completely mitigate the occurrence of illegal actions and various other problems. If a custom learning program would have been crafted this would have simplified the entire process and allowed the development of more complex programs and learning methodologies. A possible alternative would be tensor torch which is more modular and customizable and might be more suitable for this problem.

## Outlook

### Learning parameter optimization:

As mentioned in the result section there are various parameters such as learning rate, size of DQN (layers and nodes), punishment size, maximum allowed negative reward and the optimal loss function which are unknown. They have a large impact of the capability to converge and should thereby be optimally chosen. After first experimentation in this paper, it is advised that these variables are used in a super structure to find the optimal parameters which might then also be more widely used for other feeds and systems.

### Comparison to other algorithms

With RL not being the only possibility in this field, it would be useful if a comparison between the different algorithm would be shown, specifically since there are already standardized MINLP approaches which have been used traditionally for solving problems such as the sequencing one. If the convergence remains impossible, it will be interesting to see, if the efficiency for finding a high reward case is higher with reinforcement learning being included.

### Larger set of training

Once the problems in stability and runtime are resolved it would be possible to try running the training for a larger fraction of the total possibility space. The maximum in this study was around 100 000 steps which corresponds to around 0.00036% of the total space. If this were to be increased by an order of magnitude or two it would be more realistic to expect convergence.

# APPENDIX

|  |  |  |
| --- | --- | --- |
| Economic analysis | | |
| Variable | Value | Unit | Source |
| E | 90 | [%] | (Errico) |
| Y | 8000 | [h/yr] |
| G | 219.71 | [kmol/h\*m2] |
| G’ | 0.49 | [kmol/h\*m2] |
| C | 296.01 | [$/m2\*yr] |
| C’ | 17.76 | [$/m2\*yr] |
| C’’ | 20.61\*10^-3 | [$/kmol] |
| CEPCI2013 | 394 | [$] | (Laurence Midgley) |
| CEPCI2019 | 613.16 | [$] |
| Material tray | 1 |  |
| Type of Tray | 1 |  |
| linear vel. | 0.6 | [m/s] |
| Compound value if pure | | |  |
| Juli 2007, USA from Intratec.us | | |  |
| Ethane | 1,6E-05 | [$/kmol] | (Intratec) |
| Propane | 2,81E-05 | [$/kmol] |
| Propene | 4,54E-05 | [$/kmol] |
| Butane | 3,81E-05 | [$/kmol] |
| Butene | 7,54E-05 | [$/kmol] |
| Pentane | 5,76E-05 | [$/kmol] |

# References

Carran, Dorn & Hughes, John & Leslie, Alick & Kennedy, Craig. "A Short History of the Use of Lime as a Building Material Beyond Europe and North America." *International Journal of Architectural Heritage* (2012): 117-146.

CD Hubbs, C Li, Sahinidis NV, Grossmann IE, Wassick JM. "A deep reinforcement learning approach for chemical production scheduling." *Computers & Chemical Engineering* (2021).

Chan, Ching-Yao. "Advancements, prospects, and impacts of automated driving systems." *International Journal of Transportation Science and Technology, Volume 6* (2017): 208-216.

Chaves, I. D. G., López, J. R. G., Zapata, J. L. G., Robayo, A. L., & Niño, G. R. "Process Optimization in Chemical Engineering." *Process Analysis and Simulation in Chemical Engineering* (2016): 343–369.

Errico, M., Tola, G., Rong, B.-G., Demurtas, D., Turunen. "Energy saving and capital cost evaluation in distillation column sequences with a divided wall column." *Chemical Engineering Research and Design* (2009): 1649–1657.

Gani, Rafiqul. "Chemical product design: challenges and opportunities." *Computational Chemical Engineering* (2004): 2441-2457.

Gavin Towler, Ray Sinnott. *Chemical Engineering Design, Chapter-7 Capital-Cost-Estimating*. Amsterdam: Elsevier, 2013.

GeeksforGeeks. *Branch and Bound Algorithm*. https://www.geeksforgeeks.org/branch-and-bound-algorithm/. Accessed on the 15. June 2021: GeeksforGeeks, Published 2018.

Hylton, El Akil. *What is Gradient Descent*. https://medium.com/@hyltona/what-is-gradient-descent-simplified-7274ee16a8e8. Accessed on the 20. June 2021: medium.com, Created 2018.

Intratec. "Historical and Forcast data." Accessed on the 20. June 2021. Published 2017. https://www.intratec.us/chemical-markets/propane-price.

Jianyi Yang, Ican Anishchenko, Hahnbeom Park, Zhenling Peng, Sergey Ovchinnikov. "Improved protein structure prediction using predicted interresidue orientatiens." *Proceedings of the National Academy of Science* (2020): 1596-1503.

Laurence Midgley, Michael Thomson. "Reinforcement learning for chemical engineering process synthesis." Bachelor thesis. 2019.

Liddell, Scott, Jones. *Greek–English Lexicon*. Oxford: Oxford University Press, 1843.

Malone, M. F., Glinos, K., Marquez, F. E., & Douglas, J. M. "Simple, analytical criteria for the sequencing of distillation columns." *AIChE Journal* (1985): 683–689.

Miller, Steven. "Mind: How to Build a Neural Network (Part One)." Accessed on June 20. 2021. Published 2015. https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/.

Nikolaus Kriegeskorte, Tal Golan,. "Neural network models and deep learning,." *Current Biology, Volume 29, Issue 7,* (2019,): R231-R236,.

Pagmo. *Rastrigine*. Accessed on the 20. June 2021: esa, Published in 2017. https://esa.github.io/pagmo2/docs/cpp/problems/rastrigin.html.

Ribecca, Serverino. *Tree diagram*. https://datavizcatalogue.com/methods/tree\_diagram.html. Accessed on the 20. June 2021: The Data Visualisation Catalogue, unknown publishing date.

Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning*. Mountain View: Westchester Publishing Services, 2018.

Sanderson, Grant. "Neural Networks series." n.d. https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1\_67000Dx\_ZCJB-3pi.

Scott D. Barnicki, Jeffrey J. Siirola, Eastman Research Division. "Process synthesis prospective ." *Eastman Chemical Company* (2003).

Seth, C. *An Ancient Cleanser: Soap Production and Use in Antiquity*. American Chemical Society, 2015.

Stangor, Charles and Jennifer Walinga. *Introduction to psychology*. Victoria: BC Open Textbook Project, 2014.

Venkatasubramani, Venkat. "The Promise of Artificial Intelligence in Chemical Engineering." *AIChE Journal* (2018).

Xinjie Yu, Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer, 2010.

Yixin Ye, Ignacio E. Grossmann, Jose M. Pinto, Sivaraman Ramaswamy. "MINLP Model for Reliability Optimization of System Design and Maintenance Based on Markov Chain Representation." *Computer Aided Chemical Engineering, Volume 48* (2020): 1057-1062.

# Presentation

15 min online

Introduction

* ML is growing, used to solve many problems, self driving cars…. Blablabla

Goal: (Research questions)

Casestudy

Metaphor: I am a chemical engineer and I want to separate it.

* Agent starts,

Conclusion:

Agent has two choices: Feed for Column + lightkey

What is RL and how does it work?

RL = taking a complex thing and destill the essential out of that which are easy to predict

CIRCLUE OF LEARNING: AGENT(Policy)🡪 Environment

- Child doing experiments and noting its results

- Q-table: State: Flows, Actions: Feed+Light, Reward: $/h

Completed training once Q-table contains all state-action pairs.

* This results in a policy which predicts the world, no experiment needed

….. but Q table is gigantic…… how should that work?

DQN = Graph = universal function approximation (show function)

What is so good about it?

* Choose number of inputs, outputs, layer, and nodes and that it. (Simple geometry to decide)
* Very rapid to calculate outputs for any input.
* Derivate of output to weight factors can be found

Reward function

Reward = Gain- loss-Capital- Operationalcosts

Predicted reward per action – reward which was found….. is minimized == TRAINING

Lets do it. Train the DQN

Results:

* Yes it worked. Aspen can be automated, it is combinable , Vba needs to leave, Custom DQN
* What would we want? Q function vs training and then stagnates.
  + What did we get? Random non convergence

🡪Why is that? Insanely large space of combinations… only tiny part of it seen.

Two solutions: A) It has already been shown that it works with more hard coding of facts, you can reduce the space to 32 possibilities. But this then limits its use since it needs a human to make choices about the system.

B) do more training, see more about the space

* Runtime analysis
  + Aspen is limiting
  + Migley used other one, orders of magnitude faster which would mean order of magnitude more training
* How do we know it even learned anything?
  + Punishment is prevented.
  + Learn which actions result in error in aspen
  + Which actions are not possible (got better by \*10)

Conclusion:

* Aspen can be automated
* RL works on this problem,
* Problems have been identified
* Previous case: worked but simple, my case: didn’t but harder 🡪 give more computation
  + - Improvements have been identified which are needed