AI-Driven Meal Planning in the FoodTech Industry: A Reinforcement Learning Approach

Victor Mårtensson

Master's thesis
2021:E24

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Abstract

Traditional meal planning for large kitchens is a laborious and complex affair with multiple external constraints imposed on the meal plan, such as a healthy nutrition profile and a low environmental impact, which should be fulfilled while being under budget. This is a tough task for humans but by modelling the process with a Markov Decision Process and using Reinforcement Learning an agent can be taught to create meal plans from constraints. This is achieved by letting different sets of meals be represented by states while actions correspond to adding specific meals to the meal plan.

The algorithm uses an action-value function to govern the agent's behaviour through a policy. Furthermore, meal selections are rewarded through a terminal reward based on the fulfilment of the constraints. The agent is trained by generating sample episodes from following an ε-greedy policy. The return of each of these episodes is used to update the action-value function and thereby the policy. This allows the agent to learn which combinations of meals eventually can fulfil the constraints.

The behaviour of the algorithm is studied when applied to a realistic scenario. The algorithm generates a six week long meal plan of school lunches with imposed national nutritional constraints and uses existing meal data. It finds feasible meal selections which fulfil 14 constraints imposed when a sufficient data set is provided. The results illustrate that the method has good potential to be part of a new data and AI-driven approach to large scale meal planning.

Acknowledgements

This thesis was conducted in cooperation with Matilda FoodTech and I would like to thank them for the opportunity to be part of this project.

I would also like to thank both my supervisors, Alexandros Sopasakis at LTH for his guidance and suggestions through out the thesis along with Johnny Zackrisson at Matilda FoodTech for his input and effort in providing the resources and data needed for this thesis.

A thanks also goes out to Karl Åström at LTH for agreeing to examine the thesis as well as everyone at Matilda FoodTech that help in providing insight into meal planning.

Contents

Chapter 1

Introduction

Society today is highly dependant on computers, more and more tasks are automated and handled by various computer system. Vast amounts of data have been and are being collected by these systems and with increasing computational power this has lead to a surge in popularity for *Machine learning* algorithms. They thrive in this environment and is the backbone of many *intelligent* systems today as they utilise the vast amounts of data and learn from it in different ways.

The FoodTech industry is not an exception to this trend, there is an increasing awareness around the food we consume and produce. Increasing the demands on the meals served by restaurants, hospitals, elderly care facilities, schools etc. While large scale meal planning is not a new phenomenon, it is getting more and more complex all while data associated with this is accumulating. Thus leaving AI with an opportunity and potential to contribute.

This thesis is done in cooperation with *Matilda FoodTech*, a company providing software solutions for meal planning and follow ups for large scale cooking, who have gather data associated with this process. The purpose of this thesis is to investigate how a *Reinforcement Learning* model [1], can be used to create and optimise a meal plan under a set of constraints based on a data set of existing meals. It will be trained using a Monte Carlo learning approach [2].

1.1 Meal Planning

In Matilda FoodTech's own words:

Matilda is supporting millions of meals made each day across the Nordic countries, feeding our elderly and school children. A dish is designed based on a recipe, which is a combination of ingredients and a nutrition declaration. Normally, 2-3 dishes define a meal (e.g. lunch) and a set of meals are distributed over weeks or months. The meals are carefully designed to work within the constraints imposed by costs, the cooking process, equipment availability and other resources. In short, meal planning *is a laborious and expert driven enterprise.*

Meal planning today is thus mostly a human trial and error process often relaying on previous created and proven to work meal plans and accumulated knowledge in the subject. Small changes in constraints can be hard to account for and plan around when such a multitude of constraints, that can be adversarial, exist.

It is in circumstances like this, where the human ability of overview and tracking multiple objectives are failing, the power of today's computers often comes to rescue. If the problem can be stated in mathematical terms, then computer based algorithms can be utilised to track and optimise large problems under multiple constraints and objectives.

1.1.1 Meal Example From Data Set

Each data point in the data sets used correspond to a meal that has been recorded by Matilda FoodTech's software. Every meal includes a recipe with cooking instructions and proportions of all ingredients needed. Meals are often composed of subcomponents, which in turn are built of ingredients or more subcomponents, tied to the individual ingredients are nutritional values and a cost based on the amount of the ingredient specified for one portion in the recipe. Table 1.1 illustrates how the data for a meal of fried chicken with rice and sauce, originally *Stekt kyckling sås ris*, looks. This example is composed of three components chicken fillet, boiled rice and brown sauce. Some of the nutritional values of the meal plus price and weight are displayed in the table.

Stekt kyckling sås ris						
Component	Weight	Energy	Protein	Fat	Carbohydrates	Price
Kycklingfilé	107 g	596 kJ	23.4 g	5.4 g	0.0 g	4.24 kr
Ris Kokt	140 g	585 kJ	3.1 g	0.3 g	30.5 g	1.41 kr
Sås Brun	114 g	248 kJ	0.6 g	4.1 g	5.0 g	1.07 kr
Total:	361 g	1438 kJ	26.99 g	9.79 g	36.35 g	6.34 kr

Table 1.1: A meal is often composed of subcomponents, which in turn is built of ingredients or more subcomponents, tied to the individual ingredients are nutritional values and a cost based on the amount of the ingredient specified in the recipe. Here the data has been aggregated to subcomponent level and it is stated per portion, only a few nutrients of all available are specified here. The total of all subcomponents, plus a name and an id, are what makes up a data point in the used data sets. Notably, the meal lacks vegetables so it is likely this was served along side a salad buffet or similar to make it a complete meal.

1.2 The Problem

The problem this thesis tries to solve is;

Can a meal plan be created, by using a machine learning approach that takes existing meals with associated data as input alongside constraints that the meal plan should fulfil.

The constraints that will be considered are, in a descending order of importance:

1. Cost, keeping the meal plan under a fixed budget;
2. Nutrition profile, fulfilling nutrition standards;
3. CO_2-equivalent, achieving a low environmental impact.

Thus a user should be able to state the size of its meal plan, 20 meals for example, ones maximum average price i.e. the budget, nutritional values one wants achieved e.g. an average energy level of at least 2700 kJ and a maximum average CO_2-equivalent. The algorithm should then generate a feasible meal plan, made up from the provided meals, that fulfils the provided constraints and thus avoiding the laborious human trial and error approach to meal planning.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning (**RL**) is a branch of machine learning, that has probably gained most attention from its success in mastering games such as classical ones as Atari [3], board games like Go [4] as well as modern computer games such as *DotA 2* [5]. It also has many more uses such as control theory for robotics [6] and optimisations of chemical reactions [7] to mention a few.

The general framework and aim of RL is to train an *agent* to preform a task, e.g. playing a chess game, it does so by performing *actions* (chess moves) that takes it from its current *state* (the pieces position on the board) to a new state and it bases its decision on readings of its *environment* (observing the new board after opponents move). The agent thus lives in a world of finite states, for each state it has a set of actions available, it chooses an action based on some reasoning with its observations or interactions with the environment. The learning part of RL is that rewards are set for achieving desirable states (capturing the opponents queen, without losing yours) and outcomes (check mating the opponent). Negative rewards are given for undesirable states and outcomes (losing your queen without compensation, losing the game, etc.). The desired behaviour is then reinforced by letting the agent train over and over again by simulating the task, record the outcome and update its behaviour accordingly i.e. to maximise the reward. It mimics the common human learning pattern of trial and error.

2.1.1 Markov Decision Process

A very common framework for RL is that of a **Markov Decision Process** (MDP). The learning entity is called the *agent* and the system which it interacts with is called the *environment*. The agent interacts with the environment by performing *actions* and the environment responds with giving the agent *rewards* for its actions. This is often done in a discrete time setting and the agents knowledge of the environment is represented by *states*. Formally a state is notated s and it belongs to the set of all possible states, $\mathcal{S}$. In each state there is a set of possible actions, $\mathcal{A}(s)$. Thus the agent starts in state s_0 performs action a_0 gets a reward R_1 from the environment based on its action and ends up in state s_1 and so on, as can be seen in the diagram below.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{N-1}} s_N. \tag{2.1}$$

There are two types of rewards, individual rewards tied to a particular state or action and terminal rewards that is granted when the episode is terminated, i.e. a final state is reached. Both rewards make up the *return*, G,

$$G = R_1 + R_2 + \cdots + R_T. \tag{2.2}$$

The notion of *terminal states* makes most sense in applications where there is a natural start, termination and reset of the process, to some starting state, e.g. a board game. Tasks that can be framed in this manner are called *episodic tasks* and their training is divided into *episodes*, where each episode constitutes a sequence of states, from a staring one to a terminal one.

The agents goal is to maximise all rewards, i.e. the return for an episode. What the agent is searching for in reality is a *policy*, π. That should describe the agent's behaviour for any given situation by mapping all states to an action $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}(s)$. Policies can be as simple as a lookup table or a more advanced evaluation function of the agents current state in the environment.

The goal is to learn an optimal policy such that the agent maximise the return. It does so by *exploring* the state actions space of its environment. That is, it tries out different sequences of actions and updates its policy based on the return it gets. Thus the agent is learning by updating the policy in accordance with the outcome of each episode. The policy consequently represents the agent's total knowledge of its environment.

One dilemma in the learning process, is that to find the optimal policy the agent has to *explore* all possibilities to learn their value but in doing so it fails to *exploit* its current knowledge [8]. Ideally one would want the agent to exploit its knowledge, to not waste time on paths already deemed bad, while exploring just enough to not miss better paths. This makes for a natural trade off, between exploring and exploiting, in the learning process. In practice this is often dealt with by introducing stochasticity in action selection in the learning process to ensure continuous exploration.

2.1.2 Value Functions

For the agent to be able decide on actions to take, it needs a way to evaluate its current and future states. More precisely the agents policy needs to be able to assess the best action, given a state and a set of available actions. To that end a *state value function*, $V(s)$, can be utilised. The state value function describes the value of being in any given state [9] and is based on the agents knowledge. A policy then can then be to pick the action that corresponds to the next state with the greatest state value.

$$\pi(s) = \underset{a \in \mathcal{A}(s)}{\arg\max}\, V(s) \tag{2.3}$$

Alternatively, one can use a *State-Action Value function*, $Q(s,a)$ also known as *q-function*, thus giving a value to each state-action pair instead. An analogous policy is then to pick the action that corresponds to the greatest state-action value.

$$\pi(s) = \underset{a \in \mathcal{A}(s)}{\arg\max}\, Q(s,a) \tag{2.4}$$

2.1.3 Monte-Carlo Methods for Episodic Tasks

The general idea behind *Monte-Carlo learning* for episodic tasks is to learn purely from experience by interacting with the environment [1]. This is done by: a) iteratively generating an episode based on the current policy, b) evaluate the result, c) learn by updating the policy. That is the policy is updated on an episode-by-episode basis and in that respect the Monte-Carlo approach is model free as only a policy is needed to generate episodes to learn from [1].

$$V_{i+1}(a) = V_i(a) + \alpha(G - V_i(a)) \tag{2.5}$$

Constant-α MC [1], equation (2.5), $V(a)$ is an *action-value function* and $\alpha \in (0,1)$ is the *learning rate* (also called *step size*) that decides the importance of a new episode in relation to old ones. Since $\alpha = 0$ leads to no update, the previous value is just copied over, and $\alpha = 1$ means that the latest episode completely overrides the old one.

All sample episodes are generated from the current policy, if that policy is deterministic and *greedy*, i.e. it always chooses the best option, then all episodes will be the same and no further exploration will take place. One way to ensure sufficient exploration in Monte-Carlo methods is to use a *ε-greedy* policy [9]. That is a stochastic policy that with probability $(1-\varepsilon)$ follows the policy and chooses the currently believed best action and with a probability of ε it chooses a random action instead. In other words each action selected in a generated episode is: a) with a probability of ε a random action among the available ones excluding the policy choice, or b) with a probability of $(1-\varepsilon)$ the currently best action according to the policy. Thus by increasing or decreasing $\varepsilon \in (0,1)$, the level of exploration versus exploitation can be controlled.

Chapter 3

Method

3.1 Problem Setting and Framework

The framework used is that of a MDP where each state, s, represents a selection of meals and each action, a, represents which meal should be added to the selection, i.e. the meal plan and a state s_k represents a selection of k meals. A terminal state is reached when, N the number of meals in the sought meal plan, have been selected, thus s_N is always a terminal state. The value, to select each unique meal to the meal plan, is captured in the action value function $V(a)$. The action value function takes an action as input since all actions corresponds to a meal. The objective is to chose meals in a way that maximise the return.

The return, G, is only based on the terminal reward, $R_T(s_N)$, and is thus identical to it. There are no individual rewards since a selection, *meal plan*, is only judged on its total composition. For the same reason the order in which the meals are selected do not matter, only the final state s_N is of importance and it solely decides the terminal rewards and thus the return that is used to update the action value function for all actions involved to arrive at the final state. Hence the value function reflects each meal's individual capacity to contribute to fulfilling the constraints of the meal plan.

Framework summarised:

- A state $s \in \mathcal{S}$ represents a selection of meals;
- At each state an action $a \in \mathcal{A}(s)$ is selected representing which meal that is added to the selection (meal plan);
- The final state, s_N, represents a meal plan of N unique meals;
- $V_i(a_i), i = 1, 2, \ldots, N$, represents the value of selecting a meal for the meal plan;
- All meals have set of nutrition values, a cost and CO_2-eq.

3.2 Algorithm

3.2.1 General Behaviour

The algorithm's goal is to find a selection that fulfils a set of constraint. The nature of the constraints are thresholds values that one either wants to be over or under. The thresholds values for cost, nutrient levels and CO_2-equivalent per portion per meal are given as input along with a data set containing all values per portion per meal, the number of meals to select within the constraints and the hyperparameters α and ε. The algorithm operates as following it generates an episode by ε-greedily following its policy, $\pi(s)$ see (3.1), that is choosing the action with the currently highest action value, $V(a)$, among the actions available with a probability of $1-\varepsilon$ otherwise one of the other actions are chosen with uniform probability.

$$\pi(s) = \begin{cases} \arg\max_{a \in A(s)} V(a), & U(0,1) \geq \varepsilon \\ a \in \mathcal{A}(s), & \text{otherwise} \end{cases} \tag{3.1}$$

When N actions have been chosen in such fashion and the terminal state s_N has been reached, the terminal reward $R_T(s_N)$, and thus the return $G(s_N) = R_T(s_N)$, can be computed and the action value function is updated in accordance with (3.2) for all actions involved in the episode's selection, A_{sel}.

$$V_{i+1}(a) = V_i(a) + \alpha(G(s_N) - V_i(a)) \tag{3.2}$$

The action values are simply stored in an array during training and is updated by the end of each episode thus affecting the policy for the next episode. Which actions that are available at each state is tracked during each episode. The final output of the algorithm is the relative value of each action for the data set and the top N actions constitutes the proposed meal selection of the algorithm.

- The policy $\pi(s)$ (3.1) is to select $a \in \mathcal{A}(s)$ such that $V(a)$ is maximised
- The return, G, is identical to the terminal reward, R_T (3.6)
- $V(a)$ is update for all $a \in A_{sel}$ in accordance to (3.2)

3.2.2 Terminal Reward, R_T

The terminal reward is based on how well the selection fulfils a set of constrains while being under budget. All constraints are simple linear ones with a threshold value, i.e. x should be less or greater than a constant.

$$\begin{aligned} x_1 &< c_1 \\ x_2 &> c_2 \\ &\vdots \\ x_k &> c_k \end{aligned}$$

The values monitored are the average value over the whole selection in each episode, $x_k(s_N)$ the average value of the k:th constraint for the final state s_N.

$$x_k = \frac{1}{N} \sum_{a \in A_{sel}} x_{k,a} \tag{3.3}$$

The reward from each constraint are denoted, r_k, it is defined differently depending on if the corresponding constraint is a *lower* (3.4) or *upper* (3.5) limit.

$$r_k^{lower} = \begin{cases} \frac{x_k}{c_k}, & x_k \leq 1.1c_k \\ 1.1, & \text{otherwise} \end{cases} \tag{3.4}$$

$$r_k^{upper} = 1 - \frac{x_k}{c_k} \tag{3.5}$$

Lower limit constrains are hard capped at 10% above the threshold value, see (3.4), while upper limit constraints are penalised for exceeding their threshold, see (3.5). The hard cap serves to discourage large overshoots and compensation behaviour by not granting no further reward for massively exceeding the threshold.

Finally the terminal reward, $R_T(s_N)$ is constructed by the weighted sum of all r_k's with a corresponding weight w_k that is taken as input so priority of constraints can be created by rewarding selected constraints more than others. If the selection is under or equal to the budget, the sum $\sum_k w_k r_k$ constitutes the terminal reward, if the selection exceeds the budget a negative term is added to the terminal reward, see (3.6).

$$R_T(s_N) = \begin{cases} \sum_k w_k r_k, & cost \leq budget \\ \sum_k w_k (r_k - 1), & cost > budget \end{cases} \tag{3.6}$$

Thus any selection under budget will have a positive terminal reward and a selection over budget will in general lead to a negative terminal reward but if the selection is very good apart from its cost it can achieve a small positive terminal reward. That is good but too expensive selections, are rewarded more than bad and too expensive selections.

3.2.3 Alternative Terminal Rewards

Two alternative terminal rewards were developed to use as a comparison to the original one previously described. They will be denoted *triangular* and *inverse Euclidean* while the original terminal reward will be called the *regular* terminal reward when a distinction is to be made.

The triangular terminal reward, constitutes of a weighted sum in the same way as the regular terminal reward i.e.

$$R_T^{Tri}(s_N) = \sum_k w_k r_k.$$

The difference is that r_k is a triangular function centred around either the norm value, for a lower limit constraint, or half the norm value, for an upper limit constraint. See equations (3.7) and (3.8) for formal definitions.

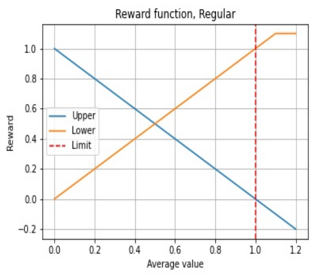
$$r_k^{lower} = \begin{cases} \frac{x_k - 0.5c_k}{0.5c_k}, & x_k < c_k \\ \frac{1.5c_k - x_k}{0.5c_k}, & x_k \geq c_k \end{cases} \tag{3.7}$$

$$r_k^{upper} = \begin{cases} \frac{x_k}{0.5c_k}, & x_k < c_k \\ \frac{c_k - x_k}{0.5c_k}, & x_k \geq c_k \end{cases} \tag{3.8}$$

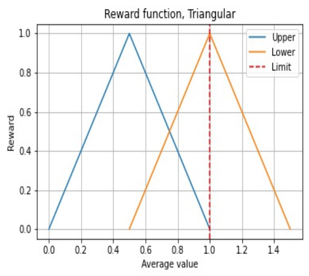
The second alternative terminal reward, is a modified inverse Euclidean distance to the sought norm. That does not distinguish between upper and lower limit constraints, as the regular or triangular terminal reward. See equation (3.9) for full definition.

$$R_T^{Euc}(s_N) = \frac{1}{\sqrt{\sum_k w_k (x_k - c_k)^2} + 1} \tag{3.9}$$

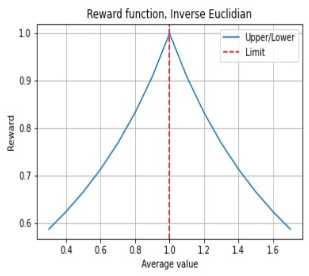
Figure 3.1 below illustrates the behaviour of all three rewards and notes some similarities and differences between them.



(a) Regular



(b) Triangular



(c) Inverse Euclidean

Figure 3.1: The reward functions considered, (a) the regular one described in the method, (b) a triangular function and (c) a modified inverse Euclidean distance. The plots shows how an individual constraint is rewarded, in relation to its norm value, depending on if it is a lower or an upper limit type of constraint. All weights are equal to one here for clarity and an easier comparison. Notably (a) and (b) distinguish between lower and upper limit constraints while (c) treats them equally. While (b) and (c) share a symmetrical property that (a) lacks.

3.3 Data Sets

The data sets used, referred to as A and B, are provided by Matilda FoodTech. They contain nutrient levels and costs per portion for a range of meals, to each meal a uniformly random CO_2-equivalent has also been added. Data set A is smaller, its prices are less accurate and it was mainly used during development of the method but it also serves as a comparison to data set B as they do not share the same source. Data set B is much larger and has accurate prices for all its meals and makes for a more realistic example.

3.3.1 Data Set A

Data set A contains around 2265 meals, they all have nutritional values per portion of the meal. The price per portion associated with each meal was incomplete and a random price has been added to those meals that lacked a price based on the available pricing data.

Constraint	μ	σ	Units
Cost	7.58	6.52	kr
Energy	1240	881	kJ
Protein	12.8	11.0	g
Fat	14.0	14.8	g
Carbohydrates	28.1	22.6	g
Fibre	3.0	2.71	g
Salt	1.66	3.52	g
Polyunsaturated Fat	1.87	2.91	g
Saturated Fat	5.18	5.62	g
Vitamin D	1.04	2.00	μg
Vitamin C	21.0	24.5	mg
Iron	1.40	1.94	mg
Folate	49.3	94.0	μg
CO_2-eq[1]	2.0	0.3	kg

Table 3.1: Means, μ, standard deviations, σ, and corresponding units for nutrient levels, cost and CO_2-equivalent per portion per meal for all meals in the data set, which contains 2265 meals in total.

3.3.2 Data Set B

Data set B contains 100 000 meals, they all have accurate nutritional values and price per portion for each meal, a random CO_2-eq has also been added to each meal. The data set is a subset of meals that have been preprocessed by removing data points that had a z-score (3.10) greater than 3 or less than -3. This was done so that extreme outliers were removed, that probably adhere from human error in logging values or fields left empty.

$$z_i = \frac{x_i - \mu}{\sigma} \tag{3.10}$$

[1]Randomly generated values.

Constraint	μ	σ	Units
Cost	10.5	6.67	kr
Energy	1691	881	kJ
Protein	19.6	10.9	g
Fat	19.9	12.8	g
Carbohydrates	40.4	23.3	g
Fibre	3.97	3.08	g
Salt	0.64	1.21	g
Polyunsaturated Fat	1.36	1.87	g
Saturated Fat	5.23	5.71	g
Vitamin D	0.85	1.0	μg
Vitamin C	24.2	24.9	mg
Iron	1.59	1.26	mg
Folate	29.4	42.0	μg
CO_2-eq[2]	1.0	0.46	kg

Table 3.2: Means, μ, standard deviations, σ, and corresponding units for nutrient levels, cost and CO_2-equivalent per portion per meal for all meals in the data set, which contains 100000 meals in total.

[2]Randomly generated values.

Chapter 4

Results

4.1 Base Case and Nutritional Norms

In this chapter a lot of different results will be presented but the core settings and parameters will mostly be the same with some isolated modifications to highlight and explore certain features. This section will define the base case used and summarise the recurring parameters and experimental setups.

The base case used to evaluated the algorithm has been that of finding a meal plan of school lunches for kids. The time frame is 6 weeks of lunches Monday to Friday, i.e. 30 meals are sought. The nutrient constraints chosen are those suggested by the Swedish Food Agency, *Livsmedelsverket*, [10] that are stated in Table 4.1.

They are national recommendations for healthy school lunches in Sweden for different age ranges. It contains specific values for 12 different nutrients that are of extra importance for children and that they recommend should be fulfilled, on approximately a 3 week average, for school lunches. The age range 10 to 12 years old have been chosen and the nutritional profile recommend for them is used to set realistic constraint on the meal plan. It is referred to as *norm 10-12* in the future. In addition an upper limit constraint on the CO_2-equivalent have been set to 0.5 or 1.0 kg. Worth noting is that for the nutrients salt and saturated fat the norm value is a highest recommended intake. Therefor they are considered upper limit constraints and are deemed successful when the average value is below the threshold. The CO_2-eq is of the same nature along with the budget.

Nutrient	10-12	16-18	Unit
Energy	2700	3300	kJ
Protein	24	29	g
Fat	24	28	g
Carbohydrates	77	94	g
Fibre	8	10	g
Salt	1.8[1]	1.8	g
Polyunsaturated Fat	5.5	6.5	g
Saturated Fat	7[1]	9	g
Vitamin D	3	3	μg
Vitamin C	15	15	mg
Iron	3.3	3.3	mg
Folate	60	60	μg

Table 4.1: Norm values taken from 'Nationella riktlinjer för måltider i skolan' [10]. It describes recommended average intake of several important nutrients for school children of age 10-12 and age 16-18 years during a school lunch. It corresponds to roughly a third of the recommended intake for a whole day.

The budget of the meal plan is in general 16 or 18 kr a high budget in relation to the costs in the data sets. The hyperparameters α and ε are always 0.1 and 0.3 respectively. In most cases the same experiment has been carried out with both data sets, with data set A 2265 meals was available and for data set B 10000 were randomly selected from the 100000 available. The number of episodes used in the training varies from 40000 and upwards. The general parameter setup, for both data sets, is summarised in Table 4.2 below.

Parameter	Value
Size of selection	30
α	0.1
ε	0.3
Norm	10-12
Budget	16 kr
Available meals	2265
Training episodes	50k - 80k

(a) Parameter setup for data set A.

Parameter	Value
Size of selection	30
α	0.1
ε	0.3
Norm	10-12
Budget	18 kr
Available meals	10000
Training episodes	40k - 100k

(b) Parameter setup for data set B.

Table 4.2: Summery of parameters used in most experiments.

The weight setup used differ between data sets and experiments more so they will be present in each section. The results will be displayed in tables summarising the average values of the final selection for each constraints along side its norm value. When it is of interest there will also be figures with plots illustrating the evolution of the selections average values during the training for picked nutrients.

[1]This is a highest recommended intake

4.2 Realistic Example

In this section, results from two good attempts of fulfilling all constraints of the base case will be shown. Firstly one attempt using data set A and secondly one attempt with data set B.

4.2.1 Data Set A

Using the base case setup and custom weights described in Table 4.3 yielded a descent result, 11 out of 14 constraint were fulfilled. Worth noting is that the price per portion 8.83 kr which in contrast to the budget of 16 kr is very good. The constraints that are not fulfilled are all upper limit constraints and the CO_2-eq limit of 1.0 kg is quite unattainable considering the mean and standard deviation of the data set (Table 3.1). The other two, salt and saturated fat, are both in the neighbourhood of their limits. All details can be found in Table 4.4 below.

Constraint	w
Protein	2
Carbohydrates	8
Fibre	2
Saturated Fat	3
Rest	1

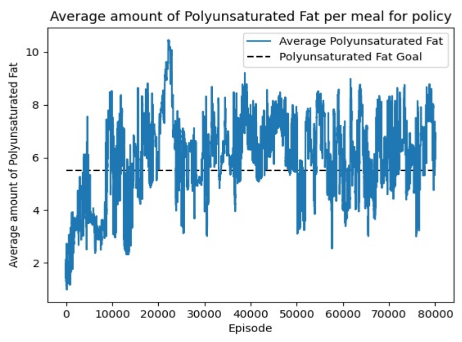
Table 4.3: Custom weight setup used for this section with data set A. Omitted constraints have the value 1.

Constraint	Norm	Algorithm	Unit
Cost	-	8.83	kr
Energy	2700	3200	kJ
Protein	24	24.1	g
Fat	24	37.1	g
Carbohydrates	77	78.7	g
Fibre	8	7.73	g
Salt	1.8[2]	1.88	g
Polyunsaturated Fat	5.5	6.73	g
Saturated Fat	7[2]	8.7	g
Vitamin D	3	3.0	μg
Vitamin C	15	42.5	mg
Iron	3.3	5.0	mg
Folate	60	200	μg
CO_2-eq	1.0	1.91	kg
Achievement Ratio	-	11/14	-

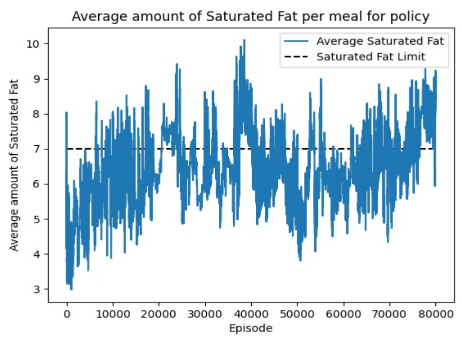
Table 4.4: Average value of each parameter considered for the final selection found by the algorithm along side the considered norm. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 16 kr, norm 10-12, a selection of size 30 and it ran for 80 000 episodes. Overall a good result, since 11 out of 14 constraints are fulfilled with only 2262 meals available. The evolution of the training can be seen in Figure 4.1.

The evolution of the average values for some of the constraints are illustrated in Figure 4.1. Of the four constraints shown in the figure only saturated fat was not fulfilled. However, as can be seen in Subplot 4.1b, that constraint is below its threshold in many instances during its training. The other three constraints all display the same general trend of increasing up to their lower limit and then oscillating around that value.

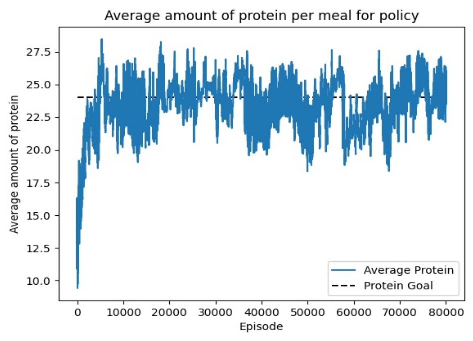
[2]This is a highest recommended intake



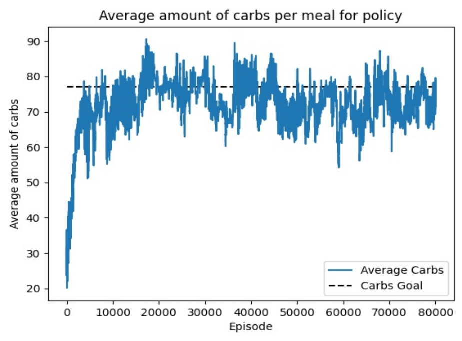
(a) Polyunsaturated fat



(b) Saturated fat



(c) Protein



(d) Carbohydrates

Figure 4.1: The evolution of the average nutrients levels of the selection during training in blue as well as each threshold value, dashed line. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 16 kr, norm 10-12, a selection of size 30 and it ran for 80 000 episodes. In (a), (b) and (d) the final values are above their lower limit, (b) however end above its upper limit. They do all illustrate an oscillating behaviour around their limits.

4.2.2 Data Set B

With data set B and a custom weight setup, see Table 4.5, a selection of 30 meals among 75 000 was obtained that fulfilled all 14 constraints.

Constraint	w
Energy	5
Carbohydrates	8
Fibre	3
Polyunsaturated Fat	3
Saturated Fat	3
Vitamin D	2
Rest	1

Table 4.5: Custom weight setup used for this section with data set B. Omitted constraints have the value 1.

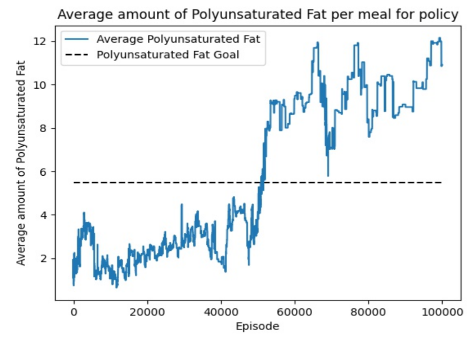
All details of the final result is captured in Table 4.6, the resulting cost per portion of the selection became 11.43 kr which is far below the high budget of 18 kr. All constraints are fulfilled and most of them with good margin.

Constraint	Norm	Algorithm	Unit
Cost	-	11.43	kr
Energy	2700	3381	kJ
Protein	24	28.97	g
Fat	24	34.55	g
Carbohydrates	77	91.38	g
Fibre	8	8.92	g
Salt	1.8[2]	0.67	g
Polyunsaturated Fat	5.5	10.9	g
Saturated Fat	7[2]	5.31	g
Vitamin D	3	3.0	μg
Vitamin C	15	20.2	mg
Iron	3.3	5.9	mg
Folate	60	100	μg
CO_2-eq	1.0	0.85	kg
Achievement Ratio	-	14/14	-

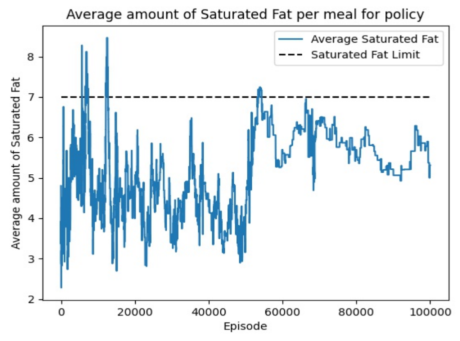
Table 4.6: Average value of each parameter considered for the final selection found by the algorithm along side the considered norm. Training was performed with hyperparameters $\alpha = 0.1$ *and* $\varepsilon = 0.3$*, a budget of 18 kr, norm 10-12, a selection of size 30, 75 000 meals were available and it ran for 100 000 episodes. The selection fulfils all 14 constraints, with good margin in all cases except vitamin D, for a average price of 11.43 kr which is way under budget. The evolution of the training can be seen in Figure 4.2.*

The evolution of some nutrient levels for the selection during training for this example can be seen in Figure 4.2. In all four examples given, the average value stays on the desired side of its limit when it is found. For the lower limit examples, shown in Subplots 4.2a, 4.2c and 4.2d, there is a clear trend of staying above the limit once it is breached. While for 4.2b, an upper bound constraint, it already starts below but on the occasions it exceeds its limit it quickly turns under the limit again.

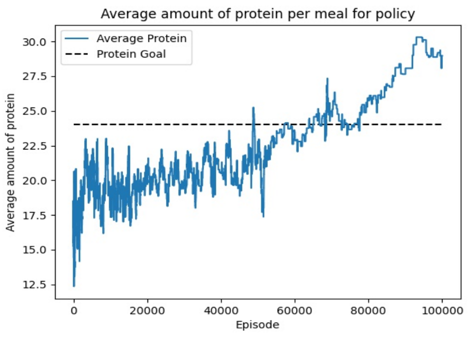
[2]This is a highest recommended intake



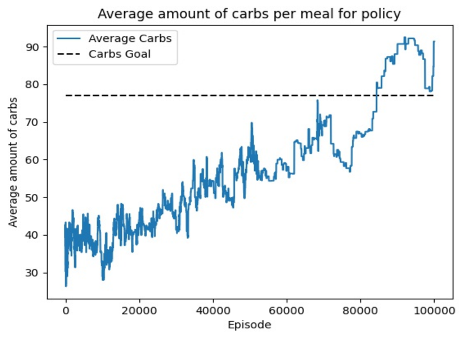
(a) Polyunsaturated fat



(b) Saturated fat



(c) Protein



(d) Carbohydrates

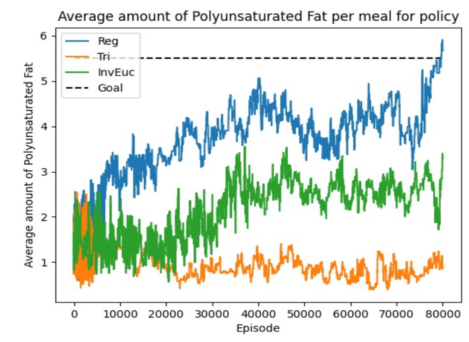
Figure 4.2: The evolution of the average nutrients levels of the selection during training in blue as well as each threshold value, dashed line. Training was performed with hyper-parameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, norm 10-12, a selection of size 30, 75 000 meals were available and it ran for 100 000 episodes. All plots illustrates the desired behaviour of the algorithm with both types of constraints. For lower limit constraints, (a), (c) and (d), the random selection's values start below their limits but they steadily increase with training episodes until they pass their thresholds. They then stays above them. For (b), an upper limit constraint, the random selection's value already starts below its limit but whenever it exceeds its limit it goes back under again.

4.3 Different Rewards

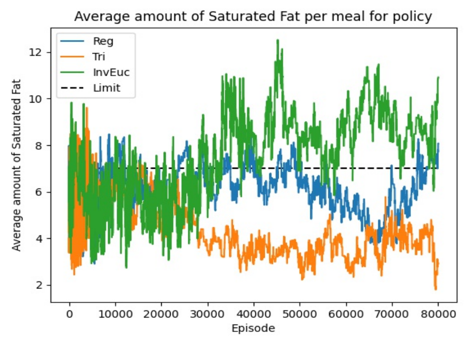
This section shows results from a comparison between the regular terminal reward function used, the triangular and the inverse Euclidean terminal reward (3.9).

All three rewards were used while training the algorithm on identical data with a identical parameter setup and number of episodes trained. Data set B was used, all weights were set to 1 and the base case with 80000 episodes were used.

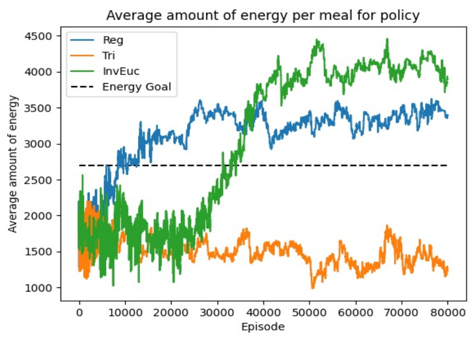
Figure 4.3 compares the evolution of the selection's average values for some nutrients between all three models. Overall the model with the regular reward performed best but for certain nutrients it was out done by the model with the inverse Euclidean distance, see Subplots 4.3c and 4.3d for example.



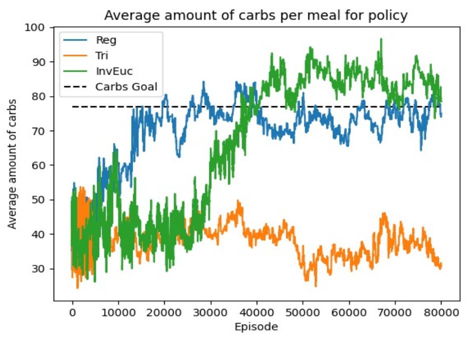
(a) Polyunsaturated fat



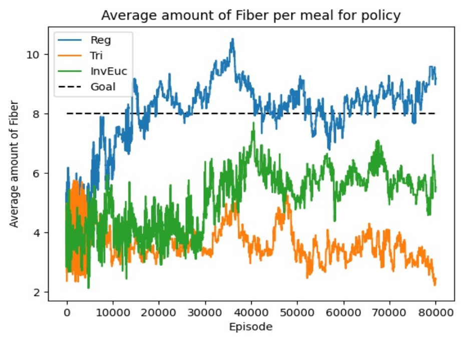
(b) Saturated fat



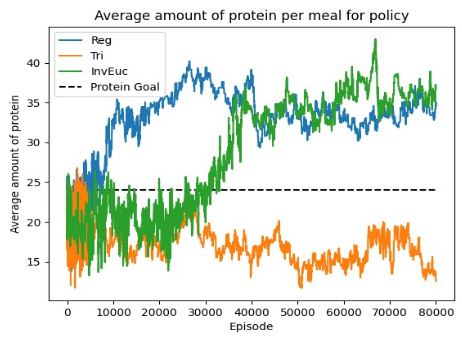
(c) Energy



(d) Carbohydrates



(e) Fiber



(f) Protein

Figure 4.3: The evolution of the average nutrients levels of the selection during training, for three different reward functions in blue, yellow and green as well as each threshold value, dashed line. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, norm 10-12, a selection of size 30, 10 000 meals were available and it ran for 80 000 episodes. Overall the regular reward performed best but the inverse Euclidean distance did not do bad, while the triangular reward was not competitive at all. Even though all three reward functions shares some properties with, one or both of the others, there are considerable difference in the outcome, suggesting that the method is quite sensitive to the reward function's construction.

The triangular reward function did considerably worse and did not compete with either of the others. While the inverse Euclidean distance was comparable to the regular one even though they differ in both symmetrical properties and in if they distinguish between upper and lower limit constraints.

Even though the triangular and the inverse Euclidean distance looks quite similar, they produced very different results and the inverse Euclidean distance was far superior. Consequently the method is deemed to be quite sensitive to the reward function.

4.4 Varying budget

In this section the results of three identical training setups with only a varying budget will be compared. This will be done with data set B and with a high, medium and low budget with two weight setups.

10000 meals were randomly selected from the 100000 meals available and used for the training. Two different weight setups, w_1 and w_2, were considered for three different budgets each. The budgets are denoted, *low*, *medium* and *high* and they corresponds to 6, 12 and 18 kr per meal respectively. These values were chosen as they reflects a bit less than average, average and a bit higher than average price per meal in the data set. Table 4.7 summaries the full set up and table 4.8 and 4.9 illustrates the resulting values for selections for both weight setups.

Constraint	w_1	w_2
Energy	1	3
Carbohydrates	1	3
Fibre	1	3
Polyunsaturated Fat	1	3
Saturated Fat	1	3
Rest	1	1

(a) Weight setups compared, w_2 has a higher weight than w_1 for five constraints. All omitted constraints have the standard weight 1.

Parameter	Value
Size of selection	30
α	0.1
ε	0.3
Norm	10-12
Available meals	10000
Budget	6, 12, 18 kr
Training episodes	40000

(b) Parameters used in this section with data set B.

Table 4.7: Experimental setup summarised.

The first weight setup, w_1 with all weights being equal to one, gave some mixed results. As can be seen in Table 4.8 medium budget got one more constraint fulfilled compared to the high budget. However looking at the individual values of each constraint reveals that the high budget selection does better than the medium budget selection in a majority of the cases. It is worth noting that the price of the low budget selection exceeded its budget of 6 kr and the price of the high budget selection is only 10.61 kr in comparison to its budget of 18 kr.

Constraint	Norm	Low	Medium	High	Unit
Cost	-	6.26	8.79	10.61	kr
Energy	2700	1672	2226	2301	kJ
Protein	24	14.91	24.42	26.9	g
Fat	24	18.48	30.69	24.02	g
Carbohydrates	77	42.8	44.6	52.91	g
Fibre	8	4.22	5.58	6.51	g
Salt	1.8[3]	0.41	0.56	0.20	g
Polyunsaturated Fat	5.5	2.09	6.32	2.97	g
Saturated Fat	7[3]	3.85	6.51	4.18	g
Vitamin D	3	1.0	5.0	3.0	μg
Vitamin C	15	21.9	41.9	26.5	mg
Iron	3.3	2.0	7.6	10.3	mg
Folate	60	100	600	700	μg
CO_2-eq	1.0	1.04	0.98	0.95	kg
Achievement ratio	-	4/14	11/14	10/14	

Table 4.8: Comparing the outcome with a low, medium and high budget. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$. We present low, medium and high budget, weight setup w_1, a selection of size 30 and norm 10-12 was used. 10 000 meals were available and it ran for 40 000 episodes. Note that the cost per meal for the low budget exceeded its threshold. There is a general trend of better results with higher budgets but there are also counter examples, notably the medium budget achieved one more constraints than the high budget.

Using the second weight setup w_2 yields similar results. But in this case the high budget selection performs better with one more constraint fulfilled than the medium budget selection, see Table 4.9. The same observations as with the first weight setup are still true. Looking at individual values, the high budget selection scores best in a majority of the constraints. As expected the low budget selection scores a lot worse than the medium and high budget selections with both weight setups.

[3]This is a highest recommended intake

Constraint	Norm	Low	Medium	High	Unit
Cost	-	5.11	9.7	12.04	kr
Energy	2700	1392	2534	2566	kJ
Protein	24	14.78	23.99	24.36	g
Fat	24	15.36	28.65	27.49	g
Carbohydrates	77	34.93	59.16	62.96	g
Fibre	8	3.39	8.55	9.45	g
Salt	1.8[3]	0.30	0.63	0.53	g
Polyunsaturated Fat	5.5	1.59	7.64	7.67	g
Saturated Fat	7[3]	3.3	4.38	4.45	g
Vitamin D	3	1.0	3.0	2.0	μg
Vitamin C	15	28.3	45.2	38.1	mg
Iron	3.3	1.7	5.4	7.1	mg
Folate	60	41	300	500	μg
CO_2-eq	1.0	0.92	1.06	0.94	kg
Achievement ratio	-	5/14	10/14	11/14	

Table 4.9: Comparing the outcome with a low, medium and high budget. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$. We present low, medium and high budget, weight setup w_2, a selection of size 30 and norm 10-12 was used. 10 000 meals were available and it ran for 40 000 episodes. There is a general trend of better results with higher budgets but there are also counter examples. The high budget case produces the best result overall with one more constraint fulfilled than the medium budget.

4.5 Comparing Weights

In this section different weight setups will be compared to study the influence it has on the solution for both data sets. This will be done by the following experiment, two selections will be obtained by training with differing weights but in all other regards an identical setup, the results will then be compared. The two sets of weights will be w_b a nominal setup with all weights equal to one and w_r a heuristic setup that has weights greater than one for some constraints. The experimental setup will differ a bit between the data sets as they differ in size and in which constraints that are hard to achieve.

4.5.1 Data Set A

For data set A the experimental setup is described in Table 4.10, the weights differ in three cases.

[3]This is a highest recommended intake

Constraint	w_b	w_r
Carbohydrates	1	5
Polyunsaturated Fat	1	5
Saturated Fat	1	5

(a) Weights for different constrains used for the blue, w_b, and the red, w_r, selection all other weights that are omitted have the value 1.

Parameter	Value
Size of selection	30
α	0.1
ε	0.3
Norm	10-12
Available meals	2265
Budget	16 kr
Training episodes	50000

(b) Parameter setup used for data set A.

Table 4.10: All weights and parameters used for the experimental setup, the results of the selections can be seen in Table 4.11.

The outcome is displayed in Table 4.11 for both the blue and the red selection, the red selection achieves better results in all three constraints that had a higher weight compared to the blue selection. However the red selection performers slightly worse than the blue selection overall as it fulfils one less constraint.

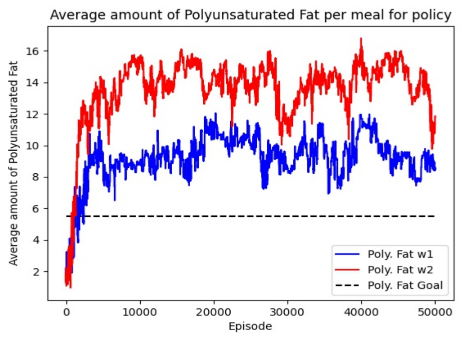
Constraint	Norm	w_b	w_r	Unit
Cost	-	10.07	9.06	kr
Energy	2700	2605	3043	kJ
Protein	24	24.73	18.62	g
Fat	24	38.89	47.03	g
Carbohydrates	77	40.93	54.7	g
Fibre	8	5.32	6.53	g
Salt	1.8 [3]	1.76	1.7	g
Polyunsaturated Fat	5.5	8.58	11.85	g
Saturated Fat	7 [3]	7.24	6.24	g
Vitamin D	3	4.0	2.0	μg
Vitamin C	15	41.9	28.3	mg
Iron	3.3	5.4	2.5	mg
Folate	60	400	100	μg
CO_2-eq	0.5	1.89	1.95	kg
Achievement ratio	-	9/14	8/14	

Table 4.11: Comparing the result with two different weight setups, w_b and w_r, all rows where the weights differed are highlighted in green. Training was performed with hyper-parameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 16 kr, a selection of size 30, norm 10-12 was used, 2265 meals were available and it ran for 50 000 episodes. The weights differed on 3 constraints in all of those, marked in green, there was a considerable improvement, indicating that weights have the desired effect and can influence the priority of constraints. However the case w_b achieves to fulfil one more constraint than w_r does and performs slightly better overall in this case.

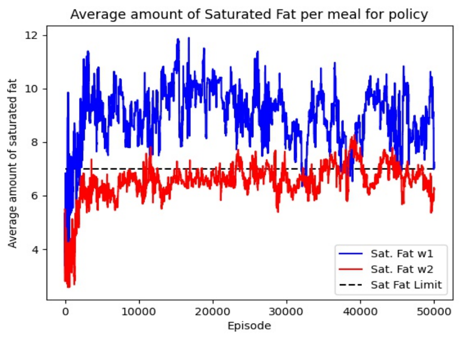
The influence of weights on the selection can be seen clearly in Figures 4.4a and 4.4b, were both types of constraint are illustrated. The red selection, which has a higher

[3]This is a highest recommended intake.

weight for the compared nutrients, has a higher average level of polyunsaturated fat and a lower average level of saturated fat than the blue selection. This is a desired outcome since polyunsaturated fat is a lower limit constraint that one wants to exceed while saturated fat is a upper limit constraint that one wants to stay below.



(a) The plot displays the average level of polyunsaturated fat for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 5 for polyunsaturated fat compare to the blue line that has a weight of 1 for polyunsaturated fat. The constrain is a lower limit one and the threshold level is marked with a dashed line.



(b) The plot displays the average level of saturated fat for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 5 for saturated fat compare to the blue line that has a weight of 1 for saturated fat. The constrain is a upper limit one and the threshold level is marked with a dashed line.

Figure 4.4: Comparing the evolution of the average value of polyunsaturated (a) and saturated (b) fat in two selections during training. Training for both was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 16 kr, a selection of size 30 and it ran for 30 000 episodes. Only difference is that the red selection has a five times higher weight for both nutrients compare to the blue selection. In both figures the red selection with the higher weights performs better as it takes higher values in (a), a lower limit constraint, and it takes lower values in (b), an upper limit constraint. Displaying that the priority of constraints can be altered with weights.

4.5.2 Data Set B

The experimental setup for data set B can be seen in Table 4.12, the weights differ here in six cases.

Constraint	w_b	w_r
Energy	1	5
Carbohydrates	1	8
Fibre	1	3
Polyunsaturated Fat	1	3
Saturated Fat	1	3
Vitamin D	1	2

(a) Weights for different constrains used for the blue selection, w_b, and the red selection, w_r, all weights that are omitted has the value 1.

Parameter	Value
Size of selection	30
α	0.1
ε	0.3
Norm	10-12
Available meals	10000
Budget	18
Training episodes	50000

(b) Parameters used in this section for data set B.

Table 4.12: All weights and parameters used for the experimental setup, the results of the selections can be seen in Table 4.13.

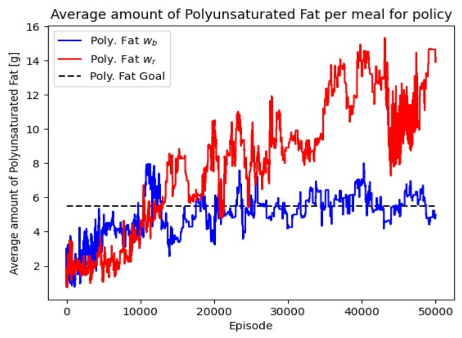
The outcome of both selections are stated in Table 4.13, the six constraint for which the weights differ have been highlighted. The desired effect is clearly shown in four of the cases, highlighted in green. In the other two cases, highlighted in red, no improvements are seen but they do not shown a much worse performance either. Since saturated fat is just slightly worse for red and neither red nor blue fulfils the constraint. Red also has a worse value for vitamin D but it is above the threshold, same as blue. Overall w_r achieves a better results as it completes 12 out of 14 constraints while blue only achieves 9 constraints.

Constraint	Norm	w_b	w_r	Unit
Cost	-	12.7	14.4	kr
Energy	2700	2367	3421	kJ
Protein	24	27.59	28.41	g
Fat	24	29.99	43.24	g
Carbohydrates	77	43.8	75.51	g
Fibre	8	6.48	11.35	g
Salt	1.8 [3]	0.79	0.52	g
Polyunsaturated Fat	5.5	4.98	14.18	g
Saturated Fat	7 [3]	7.35	7.41	g
Vitamin D	3	5.0	4.0	μg
Vitamin C	15	44.1	22.3	mg
Iron	3.3	8.5	5.2	mg
Folate	60	600	100	μg
CO_2-eq	1.0	0.95	0.91	kg
Achievement ratio	-	9/14	12/14	

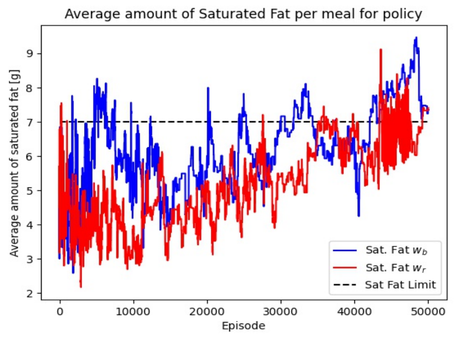
Table 4.13: Comparing the result of two different weight setups, w_b and w_r, all rows were the weights differed are highlighted in green when w_r performed better than w_b and in red if the opposite was true. Training was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, a selection of size 30, norm 10-12 was used, 10 000 meals were available and it ran for 50 000 episodes. The weights differed on 6 constraints, in four of those, marked in green, there was a considerable improvement and in two cases, marked in red, there was a slight decrease in performance. However vitamin D is above the threshold in both setups and for saturated fats it is only a minor difference in w_b's advantage. Finally as a whole the w_r setup performs better as it fulfils three more constraints than w_b.

Plots of the evolution of the selections average value of the six specially considered nutrients can be seen in Figures 4.5, 4.6 and 4.7.

[3]This is a highest recommended intake

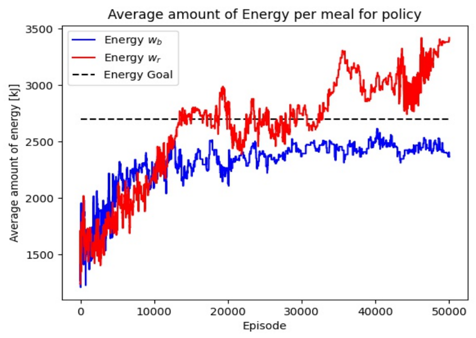


(a) The plot displays the average level of polyunsaturated fat for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 3 for polyunsaturated fat compare to the blue line that has a weight of 1 for polyunsaturated fat. The constrain is a lower limit one and the threshold level is marked with a dashed line.

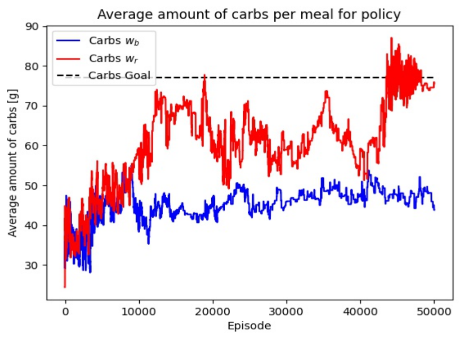


(b) The plot displays the average level of saturated fat for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 3 for saturated fat compare to the blue line that has a weight of 1 for saturated fat. The constrain is a upper limit one and the threshold level is marked with a dashed line.

Figure 4.5: Comparing the evolution of the average value of polyunsaturated (a) and saturated (b) fat in two selections during training. Training for both was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, a selection of size 30, 10 000 meals were available and it ran for 50 000 episodes. Only difference is that the red selection has a three times higher weight for both nutrients compare to the blue selection. In (a), a lower limit constraint, the red selection with higher weights performs better as it takes higher values and the red line is clearly above the blue for most parts. In (b), a upper limit constraint, it is not as clear cut, the red selection stays under the blue one for the most parts but with little margin and the final solution ends up slightly worse than the blue one. All details of the training and parameter setup can be found in Table 4.12.

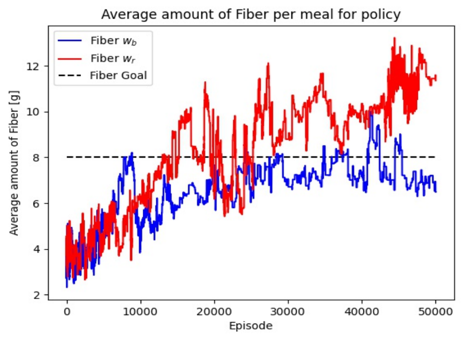


(a) The plot displays the average level of energy for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 5 for energy compare to the blue line that has a weight of 1 for energy. The constrain is a lower limit one and the threshold level is marked with a dashed line.

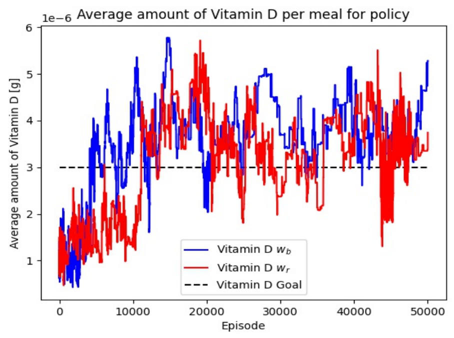


(b) The plot displays the average level of carbohydrates for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 8 for carbohydrates compare to the blue line that has a weight of 1 for carbohydrates. The constrain is a lower limit one and the threshold level is marked with a dashed line.

Figure 4.6: Comparing the evolution of the average value of energy (a) and carbohydrates (b) in two selections during training. Training for both was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, a selection of size 30, 10 000 meals were available and it ran for 50 000 episodes. Only difference is that the red selection has a higher weight for both nutrients compare to the blue selection. In both figures the red selection with higher weights performs better as it takes higher values in (a) and in (b), both lower limit constraints. There is a clear trend, in both cases, of the red selection outperforming the blue one. All details of the training and parameter setup can be found in Table 4.12.



(a) The plot displays the average level of fibre for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 3 for fibre compare to the blue line that has a weight of 1 for fibre. The constrain is a lower limit one and the threshold level is marked with a dashed line.



(b) The plot displays the average level of vitamin D for two selections, red and blue line, for each episode during training. Both runs are identical except for the weight setup. The red line has a weight of 2 for vitamin D compare to the blue line that has a weight of 1 for vitamin D. The constrain is a lower limit one and the threshold level is marked with a dashed line.

Figure 4.7: Comparing the evolution of the average value of fibre (a) and vitamin D (b) in two selections during training. Training for both was performed with hyperparameters $\alpha = 0.1$ and $\varepsilon = 0.3$, a budget of 18 kr, a selection of size 30, 10 000 meals were available and it ran for 50 000 episodes. Only difference is that the red selection has a higher weight for both nutrients compare to the blue selection. In (a) the red selection with a higher weight performs noticeably better as it mostly stays over the blue line, while in (b) there are not any clear trends but blue finishes with a higher value, both lower limit constraints. All details of the training and parameter setup can be found in Table 4.12.

Chapter 5

Discussion

The goal of this thesis is to try to create an intelligent meal planner that will take constraints as input and produce a feasible meal plan as output. This is in contrast to a traditional human approach where a meal plan first is created and just then can the prerequisite constraints be checked.

The idea was to utilise the extensive meal data that Matilda FoodTech had provided in combination with modern machine learning algorithms to solve the problem. Using a Reinforcement Learning approach with Monte-Carlo methods [2] as a foundation, we developed an algorithm to create a meal plan by selecting meals from the existing data set in such a way that the selection fulfils the specified constraints.

The constraints considered were narrowed down to:

- Cost, the selection should have an average price below budget;
- Nutritional profile, the selection should fulfil a set of nutritional constraints on average;
- Environmental impact, the selection should have an average CO_2-equivalent below a set threshold.

The algorithm works by computing a relative action value for each meal that represent each meal's individual capacity to contribute toward fulfilling the constraints of the sought meal plan. The process is modeled as a MDP, where different selections of meals are represented by states while actions correspond to adding meals to the selection.

Training is done by generating episodes by following an ε-greedy policy. The policy used always picks the action corresponding to the meal with the current highest action value. In each training episode the following steps are performed:

1. Generate an episode by following the ε-greedy policy;

2. Compute a terminal reward based on how well the selection fulfils all constraints while being under budget;
3. Updating the action value for all meals involved in the selection.

The ε-greedy approach then ensures continuously exploration through sampling while exploiting the accumulating knowledge. Updating the action value function for actions and thus also improving the policy's choice on an episodic basis.

The method is deemed successful as the results are overall good, in Section 4.2 where the best results achieved for both data sets are presented, 11 and 14 out of 14 constraints are fulfilled for data set A and B respectively. Illustrating the the method works with the larger data set, it does not achieve all constraints for the smaller data set. Although there is no guarantee that the solution had fully converged for the smaller data set. Neither is it certain that there exist a selection that fulfils all constraints in the smaller data set. For example carbohydrates, a nutrient whose constraint, 77 g, have proven to be relative hard to achieve with the data, has an average value of 40.4 g in data set B compared to 28.1 in data set A. Another example is salt, whose mean plus a standard deviation, is 1.85 g for data set B while it is 5.18 g for data set A. So within a standard deviation you achieve the upper limit of 2 g for data set B while you heavily exceed it most of the time for data set A. That is to say, the average values of nutrients are further from to the sought constraints in general for data set A.

The average values for nutrients in the selections oscillated in general quite a lot through out the training, more so for data set A (Figure 4.1) than B (Figure 4.2). Indicating that the solution might not be very stable but it could just be a natural consequence of the none deterministic nature of the method as well. On average 30% of the meals in each episode are selected randomly excluding the best option, this could be the cause for the heavy oscillations. Since throughout the training the top picks by the policy should get more cemented but the random picks can cause sudden devaluations of the policy's meals, leading to fluctuations. However there is a clear trend of increasing average values towards the limit and then a stabilisation above or around it for lower limit constrains. When considering an upper limit constraint the general behaviour is that the average level tend to return under the threshold whenever it exceeds it.

Notably many of the selections have a price per portion that is far below budget, the successful example in Section 4.2.2 for example had a price of 11.43 kr compare to its generous budget of 18 kr. The algorithm however does not reward selections more for being cheaper, it only penalise selections over budget. That is selections with identical properties except for their average price do not get rewarded more if they are 4 kr below budget compare to 1 kr below. Likewise selections do not get penalised more for exceeding the budget with 4 kr compared to 2 kr. Thus there is no incentive for the algorithm to pick the cheapest selection possible. But there is a clear pattern of relative cheap selections compare to the budget. This could possibly be an indirect effect of the policy's interaction with the terminal reward. Since for meals to stay in the top choice of the policy it is advantageous to be resistant to bad random picks and price is the biggest factor in the terminal reward. If the random picks cause a breach of the budget a large penalising term is added to the terminal reward and then used for the update of all meals involved. Consequently to stay in the policy's top picks it is favourable to be rather cheap in relation to the budget to create resistance against bad picks by being less susceptible to devaluation by random expensive sabotaging meals. Another simpler explanation could be that the general low cost in relation to the budget seen in the results, indicates that the more expensive recipes in the data used are not necessarily more nutritious.

When imposing a low budget relative to the data set, there is a big drop in performance. Around half of the constraints that gets fulfilled with a medium or high budget gets achieved with a low budget, see Table 4.8 in Section 4.4 for example. This is an intuitive outcome since a majority of the price interval is essentially removed. The same effect is not seen when comparing medium and high budgets, see Table 4.9. Where the overall performance is similar for both, that is fulfilling about the same amount of constraints. There are two obvious plausible reasons for this, one as mentioned before there are indications that more expensive meals might not necessarily be more nutritious and two the comparisons are made with the same amount of episodes of training. But with a high budget there are a lot more valid combinations available hence it might need more training to utilise that fact. But the high budget gives better results in general when comparing values for individual constraints between medium and high budget.

Using weights to manipulate the relative importance of constraints, which is illustrated in Section 4.5, worked as intended. Higher weights on certain constraints resulted in better values, this worked for both types of constraints. Also harder constraints, that otherwise did not get fulfilled, were achieved by this manipulation, see Figure 4.4 for example. But the weight setups used are specific to the data and constraints, so even though they work intuitively it can be cumbersome to find a setup that achieves the goal. As seen in Table 4.13, increasing the weight on too many constraints might not have the desired effect on all constraints either. Complex correlations between nutrients exists that might make fulfilling a combinations of goals adversarial thus leading to complex and unpredictable trade offs. Increasing a single weight too much can also lead to it being more rewarding to fulfil one constraint than several others thus having a overall negative impact.

Three different terminal rewards are compared in Section 4.3, referred to as the regular, triangular and inverse Euclidean terminal reward. They do all share properties with one another but produce varying results. The regular terminal reward does perform overall best, the triangular terminal reward does very poorly and are not competitive at all, while the inverse Euclidean terminal reward is competitive and it yields better results for some nutrients than the regular terminal reward, as seen in Figure 4.3.

Both the triangular and the inverse Euclidean terminal reward are symmetrical and rewards the distance to the sought value the same, i.e. $\pm5\%$ is rewarded the same. The triangular terminal reward distinguish between upper and lower limit constraints and centres the triangular function on the norm value for a lower limit constraints while centring in the middle of the desired interval for a upper limit constraint. The inverse Euclidean terminal reward centres on the norm value regardless of the constraints type. But while they handle the reward quite similarly, they give very different results. The triangular reward does not show a increasing or decreasing trend during training, it remains quite static around its initial values as seen in Figure 4.3. While the inverse Euclidean displays clear trends of increasing towards the norm value, see Figures 4.3c, 4.3d and 4.3f. It is the same kind of desired behaviour as the regular terminal reward is displaying. But since it does not distinguish between upper and lower limit constraints, it can end up above instead of under an upper limit, like it did for saturated fat in Figure 4.3b. So even though both alternative terminal rewards functions in similar fashion, they produce different behaviours one far more favourable than the other.

The regular terminal reward is simpler in its design. It is either an increasing or a decreasing function, depending on the type of constraint considered. This might be the reason why it functions better, it makes for an easier convergation since it does not penalise bad selections as hard. Essentially all combinations with nutrient levels above the lower limits constraints, which is the majority of constraints, are left open while both alternative terminal reward limits the interval of good choices by their symmetrical property. However that still worked well for the inverse Euclidean, so no definitive conclusion can be made of this. But it is clear that the solution of the method is sensitive to the design of the terminal reward.

5.1 Conclusions

Overall the results of the method have been positive, with the selections from the smaller data set (A), that was primary used for the development of the algorithm, fulfilling most constraints. Using the larger data set (B), selections fulfilling all constraints can be found quite easily. This is a natural consequence of the fact that data set B has, far more combinations to offer and its meals have a better potential. In other words the average values of nutrients in data set B are closer to the thresholds of the constraints, thus making it more likely that satisfying selections exist. Consequently the general notion of, better input data yields better results, holds true here as well.

Given a decent data set, such as data set B for instance, with a reasonable number of meals fulfilling the goal, it was not that hard for the algorithm to find feasible solutions. Selections fulfilling all 14 constraints have been found with only 10000 meals available and 50000 episodes of training at times. Increasing the available meals to 75 000 yielded feasible solutions most of the times at an increased cost in computational time.

Notably many of the selections have a price per portion that is far below budget even thought there is no direct incentive from the terminal reward to be more than just under budget, i.e. their is no incentive to pick the cheapest selection possible. This can be an indirect consequence of the interaction between the policy and the terminal reward or it can be an indication that the more expensive recipes in the data used do not necessarily make it a more nutritious meal.

It is a universal truth, in machine learning as well, that algorithms only learn what they are told to learn. It is crucial to find a mathematical representation that corresponds to the goal of the developer, since a computer has no notion of what a nutritious meal is. In this case the terminal reward represents the algorithms full knowledge of what a desirable selection constitutes of. Its only goal is to pick meals with the highest action value at any given instance, so it is essential that the action values represents their intended meaning. As seen here when three different but quite similar terminal rewards, designed with the same goal in mind, produced varying results that corresponded both to success and failure.

The algorithm behaves logically and has a predictable behaviour at large. Decreasing the budget made it harder to fulfil all constraints while increasing the budget made it easier, as expected. Increasing weights for certain constraints gave the intended effect but using it to optimise the selection too much are likely not worth the time investment. Since manipulating several weights leads to less predictability when the algorithm juggles the complex correlations of nutrients within meals against the constraints. Although if a clear priority of only a few constraints exist, then weight manipulation should have a plain use.

In conclusion, the general ambition has been fulfilled, the method provides a selection that adheres to the defined constraints. While the method also has a intuitive behaviour in regards to weight and budget manipulations. The algorithm can with a decent data set find selections for a realistic scenario that are:

1. Under budget;
2. Achieves all nutrient constraints, 12 considered;
3. Below average levels of CO_2-eq of the data set.

5.1.1 Outlook and Practical Considerations

The method has shown great potential for a new more automated approach to meal planning. The selection of the algorithm could serve as a first recommendation of a meal plan for many base cases and the internal ranking, that all meals get from the training process, can be used to suggest substitutions for unwanted meals. Furthermore, training on data subsets that are customised for its intended use would likely speed up training and yield better results. That is only train with meals tagged as school lunches when looking for such a meal plan, thus avoiding meals created to fit other groups needs and taste, like elderly care.

There are more dimensions to this problem than considered in this thesis. Resources needed for cooking the meals such as ovens, heated storage, staff etc imposes constraints ignored here. Time is another unaccounted factor in this. So even if the algorithm's selection fulfils all nutrient constraint and the CO_2 constraint while being under budget, there is no granite that it will work in practice since a lot of practicalities have been disregarded.

It is probably possible to include some of the resource aspects in a extended version of this terminal reward. But since it would add more complexity, it is likely more advantageous to deal with it by considering subsets of the data once again, only searching within meals that are feasible resource-wise for a particular kitchen facility. Thus trying to have those resource constraints implicitly fulfilled by the data subset.

Moreover fulfilling the desired nutritional profile lies on the assumption that the school kids, eat their planned share of a meal. In other words there is a popularity aspect here since meals need to account for the eaters preferences. A problem like this could however probably be mitigated by using data subsets intended for specific types of eaters.

Another practical concerns to the method is that it needs to retrain whenever a constraint is changed. Making it rather inflexible since training can take a long time. This could be worked around to some degree by having some pre-trained standard cases, e.g. using national recommendations for school lunches for different age ranges for a couple of budgets. An user that is planning a meal plan could then pick the closest alternative and use that meal plan as a starting point. One could also use the selected alternative as a starting point for a new training process with some modified constraints.

So the method has a big drawback in that it is sequential, and there are no obvious parallelisation to be done since each episode depends on the results of the previous episode. Otherwise this would be a standard approach to speed up the training process. Transferred learning might thus be the more natural approach to achieve a faster training process in this case.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second Edition. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[2] Sterling Osborne. "Reinforcement Learning for Meal Planning based on Meeting a Set Budget and Personal Preferences (Monte Carlo)". In: *Towards Data Science* (2018). URL: `https://towardsdatascience.com/reinforcement-learning-for-meal-planning-based-on-meeting-a-set-budget-and-personal-preferences-9624a520cce4`.

[3] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (2013). cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. URL: `http://arxiv.org/abs/1312.5602`.

[4] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–489. DOI: `https://doi.org/10.1038/nature16961`.

[5] OpenAI et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: (2019). arXiv: `1912.06680`. URL: `https://arxiv.org/abs/1912.06680`.

[6] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *International Journal of Robotics Research* 32 (2013). DOI: `https://doi.org/10.1177/0278364913495721`.

[7] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. "Optimizing Chemical Reactions with Deep Reinforcement Learning". In: *ACS Central Science* 3.12 (2017), pp. 1337–1344. DOI: `10.1021/acscentsci.7b00492`.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Third Edition. Pearson Education Limited, 2016.

[9] Marco Wiering and Martijn van Otterlo, eds. *Reinforcemnet Learning.* Springer, Berlin, Heidelberg, 2012.

[10] Livsmedeslverket. *Nationella riktlinjer för måltider i skolan.* 2019. ISBN: 978-91-7714-266-9. URL: `https://www.livsmedelsverket.se/globalassets/publikationsdatabas/broschyrer-foldrar/riktlinjer-for-maltider-i-skolan.pdf?AspxAutoDetectCookieSupport=1`.

Master's Theses in Mathematical Sciences 2021:E24
ISSN 1404-6342

LUTFMA-3443-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/