Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics & Machine Learning
2023 | LIU-IDA/LITH-EX-A--2023/001--SE

Incorporating Reinforcement Learning into Supervised Sequential Recommender Models

Patrick Siegfried Hiemsch

Supervisor : Filip Ekström Kelvinius
Examiner : Annika Tillander

External Supervisor : Sandhya Sachidanandan

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

In the context of the significant expansion of e-commerce, Recommender Systems have become important tools for businesses, enhancing customer engagement through the personalization of product recommendations. This thesis investigates the integration of Reinforcement Learning concepts into Supervised Learning frameworks, aiming to foster more accurate, novel and diverse recommendations. This study is conducted within the context of IKEA's Inspirational Feed, a feed of home-furnishing inspirations provided across IKEA's digital platforms. For this purpose, a detailed analytical comparison of three different session-based, sequential recommendation models is executed. This includes the purely supervised GRU4Rec model, as well as two hybrid approaches — SQN and SMORL — which combine Supervised Learning with the Double Q-Learning algorithm from Reinforcement Learning. The primary focus lies on SMORL, a multi-objective model explicitly designed to enhance the diversity and novelty of recommendations. As the results of this analysis reveal, all three models were able to effectively learn interrelationships among IKEA's products and Inspirational Feed images and provided reasonable next image recommendations. However, no evidence was found that the incorporation of Reinforcement Learning in the learning process helped models to improve recommendations. The thesis concludes by proposing potential directions for future research and potential modifications to the experimental design that could possibly alter these findings.

Acknowledgments

First of all, I would like to express my sincere gratitude to IKEA, the company I cooperated with for this thesis, for providing me the possibility to work on such an interesting, challenging and practically relevant topic.

Of course I also want to thank my supervisors at IKEA, Sandhya Sachidanandan and Dino Spirtovic, who took a lot of their time to answer my questions and to introduce me in detail to IKEA's Inspirational Feed, which they are both passionately working on.

I would like to extend my heartfelt thanks to my university supervisor, Filip Ekström, for his continuous support and for engaging in fruitful discussions throughout this journey. His advice and insightful tips have greatly enriched my work.

I am also grateful to my examiner, Annika Tillander, for her constructive feedback and valuable input, which has contributed to the refinement of my thesis.

I would also like to acknowledge my opponent, Omid Lavakhamseh, for his valuable insights and feedback.

Lastly, I would like to express my appreciation to Linköping University and the Division of Statistics and Machine Learning (STIMA) for offering and organizing this exceptional international master's program.

Contents

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network

CV Coverage

GRU Gated Recurrent Unit

HR Hit Ratio

IF Inspirational Feed

MC Monte Carlo

MDP Markov Decision Process

MLE maximum likelihood estimation

MSE mean squared error

NDCG Normalized Discounted Cumulative Gain

NLP Natural Language Processing

RL Reinforcement Learning

RNN Recurrent Neural Network

RS Recommender System

TD Temporal-Difference

Summary of Notation

The most frequently used symbols are presented in the following. For all other notations not included , the context needs to be taken into account and at the position they occur, their definition is provided in the accompanying text.

General Symbols

X	Input samples
$x \in X$	General instance from the input samples
$x_i \in X$	Specific instance from the input samples with index i and dimension u
Y	Target samples
$y \in Y$	General instance from the target samples
$y_i \in Y$	Specific instance from the target samples with index i
$\hat{y}$	Target prediction
C	Set of all classes in classification tasks
$c \in C$	Class included in C
v	Cardinality of C, i.e., total number of classes
E_i	Embedding matrix for sample i with column vectors $E_i^1, \ldots, E_i^u$
m	Embedding dimension (only in context of matrix E_i)
h_i^u	Hidden state of sample i at sequence position u of dimension l
θ	Model parameters
t	Timestep index
$\mathcal{L}(\cdot)$	Likelihood function
$L(\cdot)$	Loss function

Reinforcement Learning

$\mathcal{S}$	Set of all states
$\mathcal{A}$	Set of all actions
$\mathcal{R}$	Set of all rewards
S_t, A_t, R_t	Random variables for states, actions, rewards at timestep t
G_t	Cumulated discounted future reward from timestep t until terminal state T
s, s'	State and next state
a	Action
r	Reward
T	Terminal state
τ_i	Sampled trajectory with index i
γ	Discount factor
π	Policy
$\pi(a\|s)$	Probability of choosing action a in s under π
$v_\pi(s)$	Value of state s following policy π
$v_*(s)$	Value of state s following optimal policy π*
$q_\pi(s, a)$	Value of taking action a in state s following policy π
$q_*(s, a)$	Value of taking action a in state s following optimal policy π*

Glossary

Models

GRU4Rec	Supervised Learning model based on GRU-cell, used in the context of session-based next item recommendations [11]
SQN	Extension of GRU4Rec that incorporates a Reinforcement Learning part in its architecture in the form of a Q-learning based regularizer-head [39]
SMORL	Extension of SQN that adds two more Q-heads to the SQN architecture to promote the generation of more diverse and novel recommendations [29]

Metrics

HR@k	Hit Ratio for the top-k predictions, indicating accuracy of a recommendation system
NDCG@k	Normalized Discounted Cumulative Gain for the top-k predictions, closely related to HR@k, while at the same time taking into account the rank
CV_{div}@k	Diversity metric used to assess the coverage of a recommendation system over all possible next items based on the underlying dataset, taking into account the top-k recommendations
CV_{nov}@k	Novelty metric used to assess the coverage of a recommendation system over a set of unpopular items based on the underlying dataset, taking into account the top-k recommendations
R@k	Metric to approximate repetitiveness behavior of a recommendation system based on the top-k predictions

1 Introduction

1.1 Motivation

In the context of increasingly digitized commerce, Recommender Systems (RSs) represent an important tool for companies to shape their interaction with customers. By personalizing product suggestions based on customer behavior, they essentially decide which items online users discover and which they do not. Therefore, they are directly linked to companies' sales, economic success and the satisfaction of their users.

Over the years, research regarding RSs proposed different approaches for building efficient models in this area such as collaborative filtering [26], [24] or matrix factorization techniques [25], [24] based on user-item interactions. With the rise of deep learning, there has been a growing interest in developing RSs that leverage neural networks to learn complex user-item interactions and make personalized recommendations. These deep learning-based RSs have shown promising results in various domains, including e-commerce [42], social media like TikTok [20], and content recommendations e.g. on Youtube [7].

Recent literature suggests the combination of supervised sequential recommendation models, which base their predictions on a customer's most recent interaction history, with Reinforcement Learning (RL) techniques to predict the next most relevant item(s) [39], [40]. Even though these approaches show promising results, the yielded recommendations are biased towards historical patterns in user behaviour since they are trained in an offline fashion based on historic clickstream data [29]. For companies it is not only crucial to reach a high next item prediction accuracy but integrating other objectives like novelty or diversity of suggested products into RSs is equally essential, as they have been shown to directly correlate with user satisfaction scores [5], [4].

This work focuses on exploring this area by applying and comparing different RS methods to IKEA's Inspirational Feed (IF). The IF is a recently introduced feed of images with home-furnishing inspirations on IKEA's website and in their app as shown in Figure 1.1. The exact details of the IF, including the description of the specific application scenario, will be provided in the following section.

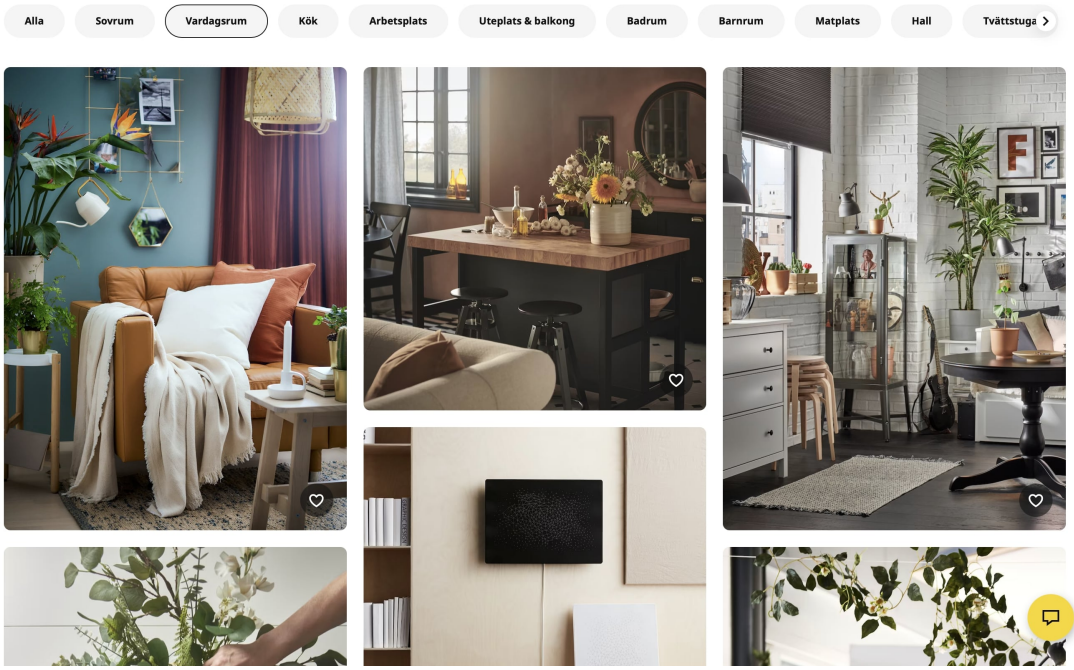


Figure 1.1: Example of the IKEA Inspirational Feed

1.2 Problem Definition

This section gives an overview of the use case provided by the collaborating company, IKEA, and how it will be approached in this work.

IKEA's core business idea is the following:

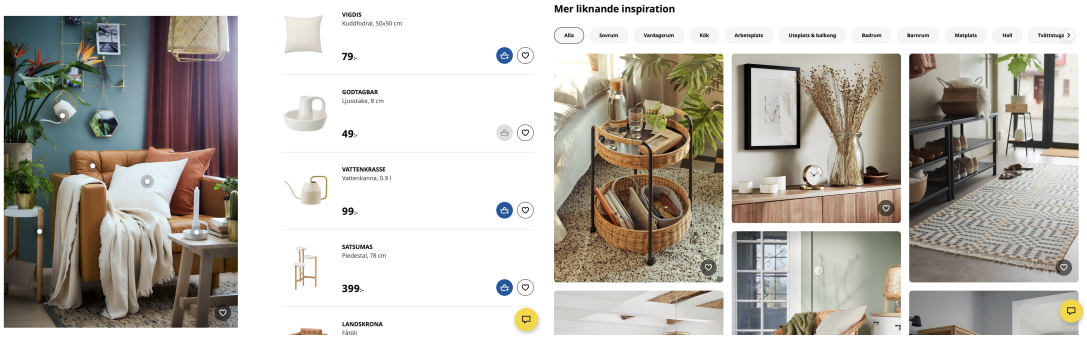
> *"To offer a wide range of well-designed, functional home furnishing products at prices so low, that as many people as possible will be able to afford them."* [1]

To bring all of their "well-designed, functional home furnishing products" to customers all around the world, IKEA runs 640 stores in 62 different markets. In these stores, rooms are arranged to showcase different living scenarios, such as living rooms, bedrooms, kitchens, and dining areas. Each room is carefully designed to demonstrate how IKEA products can be used to create practical and stylish living spaces with full-scale furniture arrangements. However, since the world becomes more and more digitized and e-commerce is playing an increasingly important role in retailing, a high-quality online presence is equally important as the physical stores.

To make the experience of interacting with their online presentation more interesting and varied, IKEA recently introduced their IF. Located both on the app and the website, the IF is a panel of clickable images with home-furnishing inspirations. The purpose is to show a composition of IKEA's products in a familiar context to spur ideas and inspiration: a kitchen table set for dinner, pillows and duvets in a bedroom, or crayons and coloring books for the children's play area. Figure 1.1 shows an example starting page of the IF with furnishing ideas for the living room area. When clicking a certain image, you will be redirected to a new page showing a larger version of the image of interest, together with a list of all IKEA products included in the picture. These products can be interacted with in different ways like viewing them, adding them to the cart, adding them to favorites, etc. Assuming that we clicked on the inspirational image on the left in Figure 1.1 showing a comfortable armchair with a pillow and some flowers, we would be directed to the page in Figure 1.2. On the left

[1] `https://about.ikea.com/en/about-us`

side we can see the top of this new website where the described product list is illustrated next to the IF image. When scrolling down further, the part of the page captured in Figure 1.2b appears, which represents a fresh batch of recommendations for new IF images that could be interesting for the customer. At the present time, twelve images are included in this batch by default. The customer can extend this by twelve further images each time they click on the "Load twelve more" button.



(a) View after image in the feed was clicked (b) New recommendation batch

Figure 1.2: Example screenshots of webpage appearing after clicking inspirational image

The new batch of recommendations is provided by an IKEA internal RS that takes in the sequence of last interactions of the customer consisting of both, products and inspirational images, to predict the new images that should be shown to the customer based on these historic interactions. Each time a customer clicks on an inspirational image, the current state of the customer, containing the last u interactions, is fed to the RS to calculate the batch of the twelve most relevant images, given the underlying RS model, to present them as new recommendations as shown in Figure 1.2b.

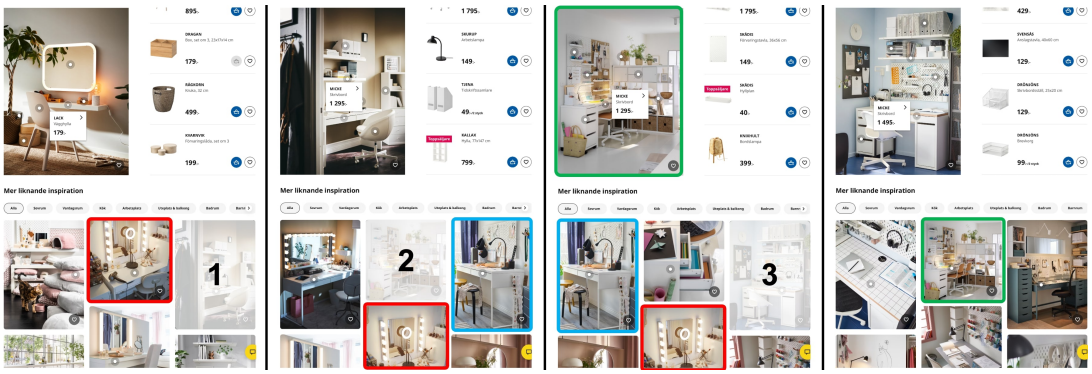


Figure 1.3: Example clickstream and according top recommendations of current RS

The current RS is working well and shows reasonable predictions for the next IF images to show to the customer given the last interacted items and images. But there is still room for improvement. When interacting with the IF on the website, it can be observed that in general, there is a high correlation between prediction batches of subsequent steps in a clickstream interaction sequence. Figure 1.3 shows an example of such a sequence of clicks in the IF (captured on 27.04.2023). The four screenshots show the page described in Figure 1.2 after clicking on a specific IF image, showing the selected picture and a new batch of recommendations. For simplicity, only the first few next image predictions out of the total twelve are included in the screenshots. The figure at the far left represents the start of the clickstream, followed by the page landed on when selecting the image marked with 1, then clicking on the recommendation marked with 2 in the next recommendation batch. The last interaction is

represented by the selection of image 3, leading to the webpage on the far right. The coloured boxes around some of the IF images are used to highlight one specific image over all four screenshots. On closer inspection, two main problems regarding the recommendations can be identified, which occur quite frequently when interacting with the current IKEA IF:

First, it is observable that the red picture showing a mirror with a ring light in front of it is included in the most relevant predictions of the RS in three subsequent recommendation batches. Also, the blue image with the white desk is in the top row of predictions for the second and third clicked IF picture. So based on this example clickstream, which can be assumed to be representative, there seems to be a high similarity among subsequent recommendation batches. Such behavior could cause boredom and dissatisfaction in the customer and represents a potential for improvement.

The second problem that can be identified is similar to the first one but poses an even greater risk of dissatisfaction. When investigating the third and fourth picture in Figure 1.3 it becomes apparent that the image clicked on and shown in the third picture marked with a green frame is repeated as a top prediction in the immediate next batch of recommendations. Since the person already clicked on this exact image before, this does not represent an interesting picture to show in the context of a feed that should inspire with new ideas and inspirations. At least not for interactions with the IF close in time to the current interaction. The repetitions marked in red and blue also pose a high risk for boredom and dissatisfaction as explained before. Nevertheless, considering that the webpage showcases twelve top recommendations, and the customer can only click on one, there is a scenario where a repetition may actually benefit the customer.

Specifically, if an item the customer was interested in, but did not select — effectively their second preference — reappears in the recommendation batch at the next time step, this could result in increased satisfaction due to the repeated exposure to a desirable item.

The green duplicates on the other hand are far more fatal: Firstly, customers are more likely to notice this type of repetition as the image was just recently clicked on. Secondly, as previously discussed, it is highly improbable that this same image will represent an appealing recommendation in the immediate future. Hence, such instances of repetition are more likely to negatively impact the customer's experience.

As already mentioned, overall the feed provides suitable recommendations and, according to IKEA, especially shows good results with regard to accuracy for next item prediction given previous customer interactions. However, as the illustrated issues show, there is still potential for improvement, especially with regard to the diversity and the repetitive character of the feed. To incorporate these goals in the training process as well, in this work the application of a multi-objective model on the given RS task is investigated and a comparative analysis with two other well-known single-objective, accuracy based approaches is conducted.

1.3 Models and Research Questions

One potential approach to address the challenges outlined in the preceding sections involves Multi-Objective models that incorporate multiple (and potentially conflicting) objectives into the learning process [29]. This thesis investigates a particular class of multi-objective model, SMORL [29], which belongs to the class of hybrid methods that combine RL and Supervised Learning. A comparative analysis is conducted with two other well-known approaches: another hybrid model, SQN [39], and the purely supervised GRU4Rec [11] model. This analysis is based on the application of the different RS methods on IKEA's IF.

The specific research questions that will be investigated in this work based on the mentioned use case are as follows:

- **RQ1:** Does incorporating a RL component within a supervised model architecture, as the case for the SQN model [39], offer the potential for improvements in accuracy of the

RS for next image prediction tasks? How will it affect the other evaluated dimensions - diversity, novelty and repetitiveness?

- **RQ2:** Does adding two more Q-heads to the RL part of the architecture from RQ1, i.e., the implementation of a Multi-objective optimization approach, help the model to further improve all (or at least some) of the evaluated dimensions - accuracy, diversity, novelty and repetitiveness - as suggested in the original work [29]?
- **RQ3:** In the context of IKEA's IF, do long-term dependencies exist in the data, and if so, does incorporating a larger window of historical interactions lead to an improvement in model performance? What is a sufficient number of previous user-item interactions needed to accurately predict the next items?

2 Theory

The aim of this chapter is to provide the reader with an overview of the theoretical concepts and foundations, which are referred to at various points in this work and which are essential in order to gain a clear understanding of the methodology employed in this work. At first, the Supervised Learning paradigm in Machine Learning is presented, including its application to regression and classification tasks. After that, an introduction to neural networks is provided, followed by an overview of the class of Recurrent Neural Network (RNN), with particular attention to the Gated Recurrent Unit (GRU). Finally an in-depth overview of the major theoretical foundations in RL is provided, followed by a presentation of the two model-free, off-policy algorithms, Q-Learning and Double Q-Learning, and their practical application in combination with neural networks.

2.1 Supervised Learning

In Machine Learning, there are three different major learning paradigms. Supervised Learning, unsupervised Learning, and RL. The choice of paradigm depends on the properties of the task at hand, with Supervised Learning used for labeled data prediction, unsupervised Learning for pattern and relationship discovery in unlabeled data, and RL for a reward driven training procedure. As the architectures in this work will use elements of both, Supervised and RL, the following section provides a brief introduction to the paradigm of Supervised Learning. All explanations are based on the theory included in the book by Goodfellow et al. [9].

As mentioned, Supervised Learning is used when labeled data is given consisting of features x and desired outputs, i.e., ground truth labels y, and the relationship of the variables is to be appropriately learned using a statistical model.

Let us assume that X stands for the body of input samples $x \in X$ and Y represent all corresponding desired output samples $y \in Y$. Additionally, it is assumed that $x, y \sim p(x, y)$, with $p(x, y)$ representing the real underlying bivariate data distribution. In the process of statistical modeling in the context of frequentist statistics, we want to find a model that picks up the relationship between x and y to make predictions about the output y when only input x is given. Essentially, we want to approximate the conditional distribution $p(y|x)$ with an

appropriate statistical model $M_\theta(y;x)$, to use it in order to provide predictions $\hat{y}$ given x, s.t.

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, M_\theta(y;x) \quad \text{where } M_\theta(y;x) \approx p(y|x).$$

The prediction $\hat{y}$ represents the most likely value for y given x and the statistical model $M_\theta(y;x)$ with parameters θ.

Two of the most common use cases in Supervised Learning are regression tasks, where the output is a continuous numerical variable, or classification, where the output is a discrete variable representing the class of an input x. In the following, the procedure of statistical modeling for each of these paradigms will be explained in more detail.

2.1.1 Regression

In regression tasks, a common practice is to assume that y is normally distributed given x, so for the conditional probability distribution $p(y|x)$ to be approximated, a normal distribution is assumed. The mean of this distribution is then modeled with a function f_θ of choice that is fit to the relationship of x and y included in the data. This function can be any function ranging from a simple linear model $f_\theta(x) = w^T x$, assuming a linear relationship between x and y, all the way to more complex non-linear models like a neural network that will be introduced in section 2.2. Given this scenario, it is assumed that

$$y|x,\theta \sim \mathcal{N}(f_\theta(x),\, \sigma^2).$$

We now want to find the values for the parameters θ of the chosen model to fit the underlying data in the best way possible. For this we will derive the likelihood function of the data given the assumed model to be able to find the set of parameters that maximizes this function. In this way, we maximize the likelihood of observing the samples $y_i \in Y$ given the observations $x_i \in X$ and the assumed statistical model. This method is called maximum likelihood estimation (MLE) . The derivation can be written as

$$p(Y|X,\theta) = \prod_{i=1}^{n} p(y_i|x_i,\theta) \tag{2.1}$$

$$= \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right) \tag{2.2}$$

$$= \frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2\right). \tag{2.3}$$

The expression in Equation 2.3 provides the likelihood function of the observed output Y given X and θ. This will now be maximized to find appropriate values for θ. We will use the log-likelihood for the derivations as it is easier and in general numerically more stable. Optimizing the logarithm of the likelihood is equivalent to optimizing the likelihood function directly because the logarithm is a monotonic function, meaning that it preserves the order of values, yielding the same result for the argmax. Since we are only interested in fitting μ at this point, modeled by $f_\theta(x)$, σ can be assumed to be a constant here. The full derivation can be found in the Appendix in section A.1.1. An excerpt from it is presented in the following:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \quad \log \mathcal{L}(\theta; Y, X, \sigma) \tag{2.4}$$

$$= \underset{\theta}{\operatorname{argmax}} \quad \log\left[\frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2\right)\right] \tag{2.5}$$

$$= \underset{\theta}{\operatorname{argmin}} \quad \sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \tag{2.6}$$

The derived expression to be minimized is the famous mean squared error (MSE) that can be used for regression tasks as a loss function for example in the context of neural network function approximators to update the parameters θ of the model in the direction of decreasing loss, i.e., increasing likelihood.

2.1.2 Classification

For classification tasks, the goal is to predict the value of a discrete variable y representing the class given the corresponding input features x.

For the following derivations, we will assume a 2-class binary classification problem. Given this assumption, in contrast to regression, where the statistical model used was the normal distribution, for binary classification the label y given the input x is assumed to be Bernoulli distributed, so

$$y|x, \theta \sim \text{Bern}(p = f_\theta(x)),$$

where p stands for the probability of the positive class $y = 1$ and $1 - p$ the probability of the negative class $y = 0$ respectively. Here, instead of the mean μ, as it was the case for the normal distribution, the distribution parameter p is approximated with a chosen function $f_\theta(x)$. Important to note is that in contrast to the approximation functions suitable for regression, which could be any function $f : x \mapsto \mathbb{R}$, here, only functions $f : x \mapsto [0, 1]$ are suitable mappings of the input. This is the case since in this binary setting the probability of the positive class is approximated, only being valid for the interval from 0 to 1.

Following the procedure for regression, we will first derive the likelihood of the class labels Y for the assumed Bernoulli-distribution given the input samples X and parameters θ. It is given by the following:

$$p(Y|X, \theta) = \prod_{i=1}^{n} f_\theta(x_i)^{y_i}(1 - f_\theta(x_i))^{(1-y_i)} \tag{2.7}$$

Accordingly, we can now take the logarithm of the expression in Equation 2.7 to derive the MLE for the parameters θ:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \quad \log \mathcal{L}(\theta; Y, X, \sigma) \tag{2.8}$$

$$= \underset{\theta}{\text{argmax}} \quad \log \left[\prod_{i=1}^{n} f_\theta(x_i)^{y_i}(1 - f_\theta(x_i))^{(1-y_i)} \right] \tag{2.9}$$

$$= \underset{\theta}{\text{argmin}} \quad -\sum_{i=1}^{n} y_i \log f_\theta(x_i) + (1 - y_i) \log(1 - f_\theta(x_i)) \tag{2.10}$$

The derived expression represents the binary Cross-Entropy loss and can be used as a loss function in combination with an appropriate function $f_\theta(x)$ meeting the criteria mentioned above to optimize and fit θ in direction of their MLE.

A common choice for f_θ, or at least for a last step before feeding the output of $f_\theta(x)$ to the loss function, is the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

which squashes values in the desired interval $[0, 1]$, making them interpretable as a probability. The input for the sigmoid can then again be any arbitrary function $f : x \mapsto \mathbb{R}$ like a linear model $w^T x + b$, as is the case for a simple logistic regression, or just the output of a linear layer from a neural network after previous other transformations of x via the layers before.

The same logic applies for the multi-class case as well, with the difference that instead of assuming a Bernoulli-distribution as for the binary case, here, the Multinoulli distribution, better know as the Categorical Distribution is appropriate. It assigns one probability to each of the classes c in the set of all possible classes C. The number of all classes is denoted with $v = |C|$ in the following. So given this scenario, it is assumed that

$$y|x, \theta \sim \text{Cat}(p = f_\theta(x)) \quad \text{where} \quad p \in \mathbb{R}^v.$$

The distribution parameter p represents the assigned class probability $p(y = c|x, \theta)$ for each class c as a vector of dimension v. Based on that, we can write out the expression for the likelihood of observing class samples in Y given the input samples X and the parameters θ:

$$p(Y|X, \theta) = \prod_{i=1}^{n} p_\theta(\hat{y} = y_i|x_i) \tag{2.11}$$

Here, $p_\theta(\hat{y} = y_i|x_i)$ is the assigned probability for the class of the real label y_i of sample i given input x_i and parameters θ of the approximation function f_θ. Practically, this will be the corresponding entry in the class probability vector yielded by $f_\theta(x_i)$. When maximizing the likelihood expression in Equation 2.11, we essentially maximize the likelihood of the observed labels given the assumed model and data. Formally, also including the log-transformation for numerical stability, the MLE of the parameters θ can be defined as follows:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \quad \sum_{i=1}^{n} \log p_\theta(\hat{y} = y_i|x_i) \tag{2.12}$$

$$= \underset{\theta}{\text{argmin}} \quad -\sum_{i=1}^{n} \log p_\theta(\hat{y} = y_i|x_i) \tag{2.13}$$

The expression in Equation 2.13 is the famous Cross-Entropy loss function commonly used in the context of classification tasks to adjust the parameters of a chosen approximation function $f_\theta(x_i)$ for p in the direction of increasing likelihood given the data. Just as for the binary case, not any function can be used. f_θ must output a valid discrete probability distribution, i.e., $\sum_{j=1}^{v} p_\theta(\hat{y} = j|x_i) = 1$ and $p_\theta(\hat{y} = j|x_i) \geqslant 0 \quad \forall j \in \{1, ..., v\}$. For the binary classification, the sigmoid function is used to ensure that f_θ meets these requirements, squashing the single output value to a valid probability. The common equivalent to this for the mutli-class case is the Softmax-function, which normalizes any vector to a valid discrete probability distribution. It is defined as follows:

$$\text{softmax}_c(x) = \frac{\exp(x_c)}{\sum_j \exp(x_j)} \tag{2.14}$$

The output is a vector of v elements, one for each class $c \in C$, containing the normalized probabilities. As input, any vector of dimension v can be used. Similar to the binary case, any function $f : x \mapsto \mathbb{R}^v$ may then be utilized as input to the Softmax function. For a neural network, this represents all transformations of x included in the layers before, with a simple last fully connected layer with v neurons, usually without activation function. The outputs of this layer that are fed to the Softmax function are assumed to represent the unnormalized log-probabilities of the different classes, often referred to as *logits*.

2.2 Introduction to Neural Networks

One of the most important models in the area of Machine Learning and the core of all developments and research efforts in the field of deep learning is the neural network, sometimes

also referred to as multilayer perceptron (MLP). This chapter aims to provide a brief introduction to this influential class of models. All of the following explanations are based on the book by Goodfellow et al. [9] except indicated otherwise. For further details, the reader is advised to refer to this source.

In general, neural networks represent flexible function approximators that can be applied to a wide range of tasks, ranging from image and speech recognition to Natural Language Processing (NLP), recommender systems and game playing. The core concept behind them can be summarized in the following equation

$$f_\theta(x) = \hat{y}$$
$$\text{s.t. } \hat{y} \approx y.$$

Here, x is the input vector that is to be mapped, y is the desired output vector and f stands for the neural network with internal parameters θ. It describes the mapping process from an input vector $x \in \mathbb{R}^n$ representing a sample from the input distribution through function f_θ representing the neural network to an output vector $\hat{y} \in \mathbb{R}^m$. The parameters θ in the network are fitted in a way that the output $\hat{y}$ approximates the real desired output y associated with input x, i.e., that $\hat{y}$ is as close as possible to y.

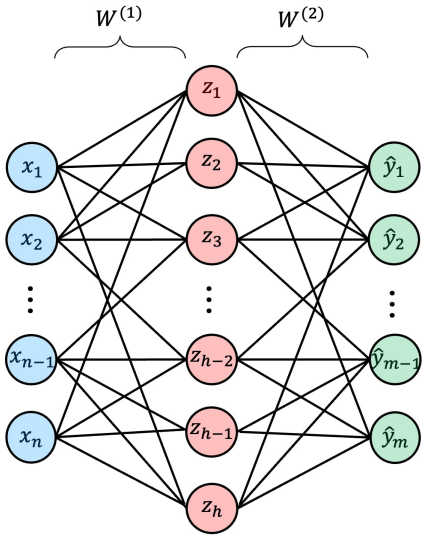


Figure 2.1: Neural network structure with one hidden layer of size h, input dimension n and output dimension m. $W^{(1)}$ and $W^{(2)}$ are the according weight matrices. Bias units are omitted for the sake of simplicity.

The architecture of an examplary neural network is provided in Figure 2.1. As we can see, it is comprised of three layers. The first one to the left is the input layer representing the input vector x. The intermediate layer is what is called a hidden layer of dimension h. The third layer on the right stands for the final mapping or prediction $\hat{y}$ of the neural network. The flow of calculations is from left to right, which is why this model class is also referred to as a *feedforward network*. Each node in the graph represents one element from the according layer vector, so x_1 for example is the first element in the input vector. Each node in a layer is connected to every node of the following layer with an associated weight. The weights represent the learnable parameters θ and are summarized in matrices for each layer. $W^{(1)}$ is the first weight matrix and is of dimension $(n \times h)$ and $W^{(2)}$ stands for the second one with dimension $(h \times m)$. The values of the nodes in subsequent layers are calculated as linear combinations of weights and nodes of the previous layer. In the operations in the hidden layer and, if desired, also in the output layer, it is common to have an additional non-linear activation function which is applied to the computed values. This is crucial to introduce

nonlinearity in the mapping process, providing the model with the capability of fitting not only linear but non-linear functions as well. The given structure can easily be extended by adding further hidden layers creating a deeper and more complex architecture. But as the Universal Approximation Theorem states, "*multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available*" [13].

Assuming the architecture in Figure 2.1, the computations for the different layers can be written as follows:

$$z = \sigma(W_1^T x + b) \tag{2.15}$$

$$\hat{y} = W_2^T z + c \tag{2.16}$$

Here, the notation for $W^{(i)}$ was changed to W_i for a cleaner expression. The vectors $b \in \mathbb{R}^h$ and $c \in \mathbb{R}^m$ are the bias weights which were omitted in Figure 2.1. σ stands for the activation function used in the hidden layer and could be one of many alternatives like the Rectified Linear Unit (ReLU) or the Hyperbolic Tangent (tanh). In the equation, no activation function for the output layer was considered. The full mapping from x to $\hat{y}$ by the network is defined by

$$f(x; W_1, b, W_2, c) = W_2^T \sigma(W_1^T x + b) + c. \tag{2.17}$$

The weight matrices as well as the bias vectors are the set of parameters of the neural network and they are trained in a way that the resulting function approximates a given function as close as possible. The function itself is represented by a body of sample inputs $x \in X$ with corresponding desired mappings $y \in Y$ from the real underlying distribution. The optimization process of fitting the parameters to the given data is in its core based on the concept of gradient descent, essentially adjusting the weights according to the direction indicated by their gradient based on a predefined loss function which shall be optimized. One of the most common choices for the loss function in the context of regression tasks is the MSE loss. It is used to fit $\hat{y}$ as close to y as possible, representing the MLE of the parameters given the assumption of normally distributed errors as introduced in Chapter 2.1.1. For classificatin tasks, the Cross-Entropy function is commonly utilized. For instance, when training a neural network on a classification task, the Cross-Entropy loss from Equation 2.13 will be minimized, getting as input the output-logits from the final layer normalized to a probability distribution by softmaxing.

To compute the gradient of the different parameters given the underlying loss definition, the backpropagation algorithm is used. It utilizes the chain rule of differentiation to efficiently calculate the gradient for each weight in the different layers to be able to then take a step in the direction that minimizes the loss function. For the given network in Figure 2.1 the gradient for the weights $W^{(1)}$ can be written as follows, assuming an arbitrary loss function $L(\hat{y}, y)$ with $\hat{y} = f_\theta(x)$:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W^{(1)}} \tag{2.18}$$

The given equation in 2.18 is only shown for a single sample x, y but can easily be extended to multiple data points - a batch - e.g. by taking the average of the gradient over these samples, approximating the expected value of the gradient based on the given data. Taking the most basic formulation of the gradient descent algorithm as an example, one weight update in the direction of decreasing the loss (i.e., opposite direction of the gradient) can be written as follows:

$$W^{(1)} = W^{(1)} - \alpha \frac{\partial L}{\partial W^{(1)}} \tag{2.19}$$

One more interesting thing to note in the context of classification is that when computing the gradient learning signal for the combination of Cross-Entropy loss together with Softmax based on a single sample x_i from the training set belonging to class $y_i = c$, this signal is not only limited to the neuron in the output layer corresponding to class c. However, since all class logits are included in the normalization, a gradient signal is flowing to the units for all classes. This signal will be positive for the real class label y, pushing up the corresponding logit in the output layer and negative for all other classes, pushing their logits down, all contributing to increasing the outputted probability for the real class label from the Softmax layer.

For more details on the optimization procedure, algorithms and loss functions, the reader is referred to the mentioned source [9].

2.3 Recurrent Neural Networks

In the previous section, the fundamental concepts of neural networks were presented. In this section, the focus will be on a specific type of neural network known as Recurrent Neural Networks (RNNs). All of the following explanations are based on the books by Goodfellow et al. [9] and Géron [8] except indicated otherwise. For further details the reader is also advised to refer to these sources.

Traditional feedforward neural networks process information in one direction, from the input layer through the hidden layers to the output layer. However, for certain tasks such as NLP or time series prediction, the sequence of inputs needs to be considered. RNNs are a class of neural network specifically designed for this purpose.

RNNs are characterized by their ability to process sequential data by maintaining a hidden state that is updated at each time step and is fed to the network itself for the computations in the next time step. The hidden state serves as a sort of "memory" that enables the network to remember information from previous time steps and uses it to process the input at the current time step.

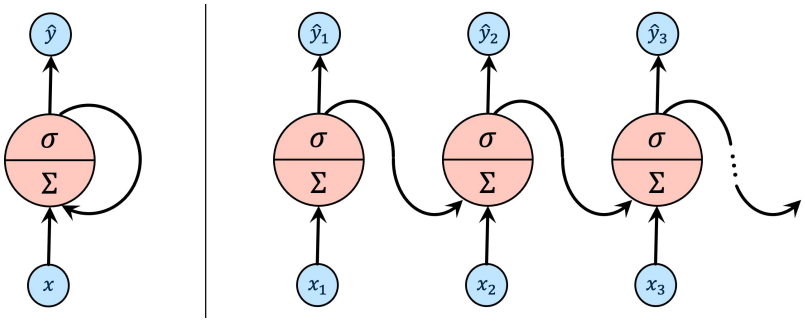


Figure 2.2: Single RNN neuron (left) and the unrolled version over the first three steps in a sequence (right). Adapted from [8].

Figure 2.2 shows a simple RNN neuron on the left, where the input vector x gets processed, first by calculating the weighted sum indicated by Σ and then by applying an activation function σ to the outputs, yielding the output $\hat{y}$. So far, this is the same behavior as what happens in a normal neural network neuron, for example in the hidden layer of the architecture shown in 2.1. But in contrast to this, in the RNN neuron, there is another connection leading back into the input, essentially feeding the output from the step before into the computation of the output for the current time step. This becomes more apparent in the right visualization in Figure 2.2 which shows how the one unit RNN cell gets unrolled over

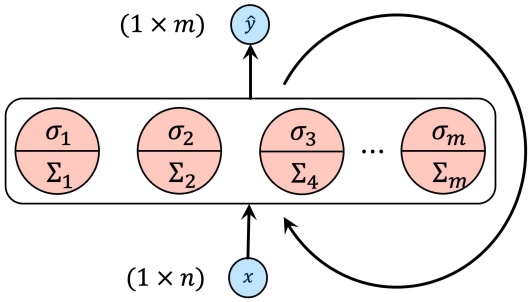


Figure 2.3: Simple RNN cell with m neurons yielding an output $\hat{y}$ of dimension m. Adapted from [8]

a sequence of inputs, first processing one timestep and then feeding its output to the next cell where, together with the new input, x gets mapped to the output utilizing the weights connected to the unit. It is important to mention that these weights are entirely shared across steps in a sequence. Therefore, the same neurons, i.e., the same weight matrices are used to compute the outputs at all timesteps. Given this simple model with just one neuron, the input vector $x \in \mathbb{R}^n$ will be mapped to just a single number $\hat{y}$. This architecture can be easily adapted to output a vector of arbitrary dimension by adding more neurons to the RNN cell. A rough overview of a cell with m units can be found in Figure 2.3

To compute the output $\hat{y}_t$ at timestep t, the following calculations need to be followed:

$$\hat{y}_t = \sigma(W_x^T x_t + W_y^T \hat{y}_{t-1} + b) \tag{2.20}$$

It is observable that except for the additional input of $\hat{y}_{t-1}$, the equation is similar to the usual way how the activations in a normal layer are computed, as described in Equation 2.15. Here, $W_x \in \mathbb{R}^{n \times m}$ stands for the weight matrix containing the parameters for the weighted sum of all n entries in x_t for each of the m neurons. b is again the bias and has dimension m, i.e., one bias per unit. The output $\hat{y}_{t-1}$ from the previous timestep enters the equation as well, with a separate set of weights $W_y \in \mathbb{R}^{m \times m}$. The output for the presented architecture would be the vector $\hat{y}_t$ of dimension m.

The parameters of the model can be trained similar to the basic neural network by computing the gradient with respect to all weights and biases and using it to upgrade them with some sort of gradient descent variant as shown in Equation 2.19. However, for the RNNs the backpropagation algorithm needs to be applied on the fully unrolled network, which is then called *backpropagation through time*. The following equations illustrate the unrolling of the network in its recursive nature:

$$\hat{y}_t = \sigma(W_x^T x_t + W_y^T \hat{y}_{t-1} + b) \tag{2.21}$$
$$\hat{y}_{t-1} = \sigma(W_x^T x_{t-1} + W_y^T \hat{y}_{t-2} + b) \tag{2.22}$$
$$\vdots$$
$$\hat{y}_1 = \sigma(W_x^T x_1 + b) \tag{2.23}$$

In Equation 2.23, which represents the computations at the first step of a sequence, the initial value for the previous input is assumed to be zero for simplicity. To compute the gradient of the loss function at timestep t, the whole recursive path of the function along all t steps needs to be considered.

The discussed architecture represents the most basic RNN model, where the output $\hat{y}$ equals the vector of result from the neurons in the cell, thus being of the same dimension m as the number of units included in the cell. But this can be adapted to a more general concept,

where the output vector differs from the vector computed by the RNN neurons. This vector is in the following referred to as hidden state h. This hidden state is used to calculate the output $\hat{y}$ and is then passed to the cell for the computations in the next timestep. The altered equations are illustrated in the following:

$$h_t = \sigma(W_x^T x_t + W_h^T h_{t-1} + b) \tag{2.24}$$

$$\hat{y}_t = W_y^T h_t + c \tag{2.25}$$

Here, $W_h \in \mathbb{R}^{m \times m}$ is the weight matrix to process the previous hidden state h_{t-1} and $W_y \in \mathbb{R}^{m \times l}$ holds the parameters to map the current hidden state h_t to the desired output dimension l of $\hat{y}_t$. c is the bias for the output, also of dimension l.

In general, RNNs can be used for a variety of different problem settings that can broadly categorized into the following types:

1. **Sequence-to-Vector (Encoder):** An input sequence is encoded into a vector representation of fixed size. The easiest way to model this is by feeding the sequence to an RNN and returning the last hidden state after processing the last element in the sequence.

2. **Vector-to-Sequence (Decoder):** Taking a fixed sized vector as an input and translating it to a sequence of outputs.

3. **Sequence-to-Sequence (Encoder-Decoder):** Common problem structure found in NLP language translation tasks. Takes a sequence as an input and likewise returns an output sequence.

2.3.1 Gated Recurrent Unit

As shown in the previous section, RNNs have proven to be effective in modeling sequential data due to their ability to capture temporal dependencies. However, the introduced standard RNN model architectures often show unstable training characteristics in practice. This is due to the fact that, especially when processing long sequences, the recursive function graph introduced in Equations 2.21 to 2.23 required for computing the gradients for the parameter updates gets very deep. Similar to very deep standard neural network architectures, this can lead to stability problems. Therefore RNNs often suffer from vanishing gradients, where gradients propagated over many time steps can become very small, making it difficult for the model to learn long-term dependencies and update weights in an appropriate way. In the same way, this can also lead to exploding gradients, where gradients become too large and result in unstable learning behavior.

To address these issues, carefully designed models like the Long Short Term Memory (LSTM) [12] or the GRU [6] were introduced as a type of RNN architecture that incorporates gating mechanisms to control the flow of information within the network. The main motivation behind their invention was to enable RNNs to capture long-term dependencies more effectively and to address the issue of vanishing gradients.

The GRU, first introduced by Cho et al. [6] in 2014, represents a simplified and slimmed variant of the LSTM, showing comparable results on many tasks while at the same time having less parameters, thus reducing model complexity. Since the architectures underlying this work will exclusively use GRU layers, only this model class will be presented in the following.

Figure 2.4 shows an inside view of the GRU cell. We can see that the model receives two inputs, the previous hidden state h_{t-1} from the cell associated to the timestep before and the new sequence element x_t at the current step t. These two vectors are processed within the cell and finally transformed into the new hidden state h_t, which is passed to the computations in the next timestep as well as passed as an output for timestep t. Therefore, the cell can

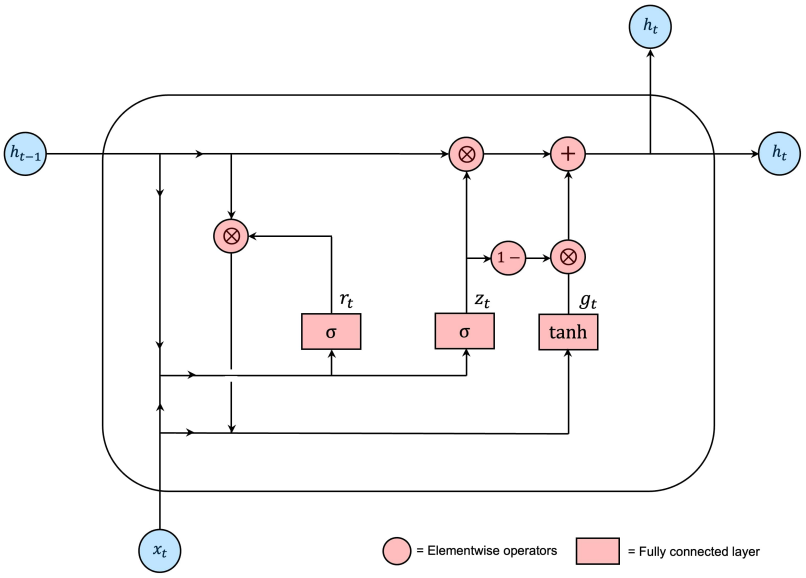


Figure 2.4: Visualization of GRU cell (adapted from [8])

be summarized as the mapping $(x_t, h_{t-1}) \mapsto f_\theta(x_t, h_{t-1}) = h_t$. To remain consistent with the previous formulations, it is assumed that $x \in \mathbb{R}^n$ and $h \in \mathbb{R}^m$. To get a prediction $\hat{y}_t$ for t, the yielded hidden state can be processed further, for example by using a simple fully connected layer without activation function to transform h_t as illustrated in Equation 2.25. The following equations illustrate the visualized transformations inside the GRU cell from x_t and h_{t-1} to h_t

$$z_t = \sigma\left(W_z^T x_t + U_z^T h_{t-1} + b_z\right) \tag{2.26}$$

$$r_t = \sigma\left(W_r^T x_t + U_r^T h_{t-1} + b_r\right) \tag{2.27}$$

$$\tilde{h}_t = \tanh\left(W_h^T x_t + U_h^T (r \otimes h_{t-1})) + b_z\right) \tag{2.28}$$

$$h_t = z \otimes h_{t-1} + (1 - z) \otimes \tilde{h}_t. \tag{2.29}$$

Inside the GRU cell there are three different fully connected layers, each one representing one of the intermediate results for the final calculation of h_t in Equation 2.29. As a consequence of the presented formulas, especially given the numerous element-wise operations, the number of neurons in each of these layers is fixed to the dimension of h_t. There are two gates included in the cell, the *update gate* represented by z_t in Equation 2.26 and the *reset gate* r_t shown in Equation 2.27, which are both calculated in the same way just with different underlying weight matrices. The weights transforming the input x_t are W_z and W_r, both of dimensions $(n \times m)$. The weights interacting with h_{t-1}, U_z and U_r, have the dimensions $(m \times m)$. b_z and b_r stand for the m-element bias vectors. Both gates apply the sigmoid activation function, represented in the equations as σ, to the final vector, ensuring all entries to lie in the interval $(0, 1)$. Together, these gates allow the GRU to selectively update its hidden state based on both the previous hidden state and the new input.

The update gate z_t controls how much of the previous hidden state should be kept in the hidden state of the current time step and how much of the new candidate hidden state $\tilde{h}_t$ should be taken in, given the current input x_t and previous state h_{t-1}. If the value for z_t is close to 1, the gate is closed and the entire old hidden state - representing the "memory" of the model - is kept. If it is close to 0, this means that the old hidden state is entirely overwritten

by the new candidate $\tilde{h}_t$ - the gate is open. In essence, the new hidden state h_t represents a mixture of the old state h_{t-1} and the new candidate $\tilde{h}_t$, as all values in z fall in $(0, 1)$. It is important to mention, however, that this process of blending h_{t-1} and $\tilde{h}_t$ happens on an element-wise level, as z is not a scalar but rather a vector. So some entries of the old hidden state h_{t-1} might be kept, while others are replaced by or blended with the entries from $\tilde{h}_t$.

The reset gate r_t works in the exact same way. It determines which parts of the old hidden state h_{t-1} are relevant given x_t and h_{t-1} and should be included in the calculations for the new candidate state vector $\tilde{h}_t$. If the element in the gate r_t is close to 1, the corresponding element in the old state vector h_{t-1} is considered in the calculations of $\tilde{h}_t$. If the gate is 0, it will be excluded from computations, i.e., reset.

As already mentioned, $\tilde{h}_t$ represents a candidate vector - a candidate for becoming the new hidden state of the RNN cell at timestep t. It is calculated analogously to the gates through a fully connected layer with two weight matrices W_h and U_h and the bias b_z. But here, h_{t-1} first get transformed by the reset gate r_t, as explained before. As an activation function not the sigmoid but the tanh function is used instead, squashing the elements of the candidate state to the interval $(-1, 1)$.

This specialized architecture with all its illustrated properties and carefully designed gating mechanisms helps to mitigate the vanishing or exploding gradient issues of basic RNN models, allowing for a more stable training process that can more easily pick up long-range dependencies included in the data, possibly resulting in a better performance and generalization of the model.

2.4 Reinforcement Learning

RL is one of the three major learning paradigms used in Machine Learning. In comparison to Supervised Learning, RL systems are trained in a reward-driven fashion by interacting with an environment to essentially learn good decision strategies that maximize the observed rewards. As the name suggests, training is performed by reinforcing high reward behaviors while avoiding low reward strategies. Especially in recent years, RL has gained more and more popularity in research as well as in a business context, not least because of the famous breakthroughs that companies like Google DeepMind[1] have achieved. For example, DeepMind developed an algorithm called AlphaGo [27] in 2016 that defeated the world champion in the game of Go, which was particularly surprising, since the old Chinese game is known for its complexity and the vast number of possible moves. Another very recent technique mostly used in NLP is called Reinforcement Learning from Human Feedback (RLHF) [22]. It can be used to efficiently fine tune large language models like ChatGPT[2] with feedback from humans to further optimize the outputs in a desired direction.

The following section provides a general introduction to RL, presenting the major theoretical foundations leading to the specific algorithms, Q-Learning and Double Q-Learning, that are used in the models underlying this thesis. After that, we will take a look at how function approximation in the form of neural networks can be integrated into the learning process. In this context, two important algorithms in this field, Deep Q-Networks (DQN) and Double DQN (DDQN) will be briefly presented as well. All of the following explanations are based on the book by Sutton and Barto [30] from 2018, except indicated otherwise.

2.4.1 Markov Decision Process (MDP)

RL represents a reward-driven learning procedure in which an agent learns to interact with an environment via actions in order to maximize a cumulative reward signal. The theoretical framework underlying this class of problems is the Markov Decision Process (MDP). An

[1] https://www.deepmind.com
[2] https://openai.com/blog/chatgpt

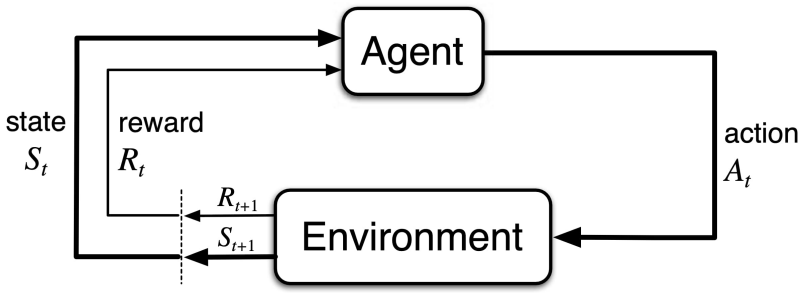


Figure 2.5: Overview of the problem modeled by an MDP [30]

overview of the general process and its core components can be found in Figure 2.5. The easiest way to explain the dynamics and elements of an MDP is to use a game like Tic-Tac-Toe[3] or the Atari arcade classic "Breakout"[4] as an example, where the player controls a small platform to bounce a ball and break through a wall of bricks. Possible movements of the platform are only left and right. In an MDP, the agent represents the entity taking decisions about actions inside a specific environment at a certain point t in time. For a game, the agent would represent the player inside an environment represented by the game or level structure itself. In Breakout, at each timestep t, the player, i.e., the agent, takes an action $A_t \in \mathcal{A}$ with $\mathcal{A}$ representing the set of all possible actions. This decision influences its state $S_t \in \mathcal{S}$, leading it to transition to the following next state $S_{t+1} \in \mathcal{S}$. Hereby, the state corresponds to a numerical representation of the environments state, ideally containing relevant information for deciding on the next action A_{t+1} to take, given this new state S_{t+1}. For Tic-Tac-Toe, the state could simply be the position of circles and crosses on the 3x3 game grid. In Breakout, the state could contain information like the current position of the platform, of the ball, which walls are already broken, etc[5]. So only given the information contained in its current state S_t, the agent decides on action A_t to take, to which the environment reacts with a new agent state S_{t+1}. For Tic-Tac-Toe for example, the new state would contain new information about how the other player reacted to the agents move A_t at timestep t, so where they placed their character on the grid. In addition to the state S_{t+1}, the environment also provides a numerical reward signal $R_{t+1} \in \mathcal{R}$ with $\mathcal{R} \subset \mathbb{R}$, representing a measure for fail or success of a given (sub)goal. In general, a positive value is a rewarding feedback, indicating that something desirable happened in the environment. A negative signal on the other hand represents a sort of "punishment", indicating undesirable changes with regard to the set goal(s). The exact reward structure is a design choice and often a result from experimenting. For Tic-Tac-Toe this could for example just be an indicator whether the game was won or lost, so trying to optimize for the end goal of winning. For Breakout, the player receives consecutive points increasing the total score when walls were successfully broken after a move, which could be used as reward signals as well, providing a stream of rewards rather than just one terminal one. The actions A_t together with the states S_t and rewards R_t form a sequence ordered by their time t of occurrence, also called a trajectory τ_i, which can be represented as

$$\tau_i = (S_0, A_0, R_1, S_1, A_1, ..., A_n, R_n, S_n),$$

with S_n as the final state marking the end of the episode. Note however, that not every problem in RL must have terminal states, but can entirely be defined by one long, ongoing trajectory called *continuing task*.

Most problems in RL will be formulated as a finite MDP where all sets $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ are finite, so they are discrete in nature and only contain a certain, finite number of elements.

[3] https://en.wikipedia.org/wiki/Tic-tac-toe

[4] https://en.wikipedia.org/wiki/Breakout_(video_game)

[5] In practice, just multiple consecutive screenshots from the game are taken as high dimensional inputs to represent states.

One of the central characteristics of an MDP is the assumption of the *Markov Property*. It means that future states S_{t+1} and rewards R_{t+1} only depend on the direct predecessor state S_t and the action A_t chosen in this state. Given S_t and A_t, S_{t+1} and R_{t+1} are independent of any states or actions further in the past. As David Silver, a leading research scientist from DeepMind focused on RL, describes in one of his lectures:

"The future is independent of the past given the present"[6]

Applied on the small example of Tic-Tac-Toe, given the state of the grid, so the position of all crosses and circles on the board, other information about the past are irrelevant. Thus, this property forces the state to contain all necessary information to completely characterize the underlying process. Given these characteristics of the MDP, the dynamics of it can be formulated with a conditional joint probability distribution $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ as

$$p(s', r|s, a) = P(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a),$$

where the lowercase letters stand for specific values that the corresponding random variable takes, and with s' describing the next state. Assuming a finite MDP, $p(s', r|s, a)$ is a discrete probability distribution with the following properties:

$$0 \leqslant p(s', r|s, a) \leqslant 1 \quad \forall s', s \in \mathcal{S}, \forall r \in \mathcal{R}, \forall a \in \mathcal{A} \tag{2.30}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.31}$$

Given this definition, the Markov Property can be formally defined as

$$P(S_{t+1}, R_{t+1}|S_0, A_0, S_1, A_1, ..., S_t, A_t) = P(S_{t+1}, R_{t+1}|S_t, A_t).$$

In summary, the MDP in RL is defined as a 5-tuple of $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$, with γ as discount factor and p as the dynamics of the environment [23]. More details on γ are introduced in the next section. With the MDP as a core theoretical backbone of RL, a wide variety of different problems can be modeled as such.

2.4.2 Reward

As mentioned before, RL represents a reward driven learning procedure with the general goal of training an agent that maximizes the expected discounted future reward when interacting with the environment. Most RL problems are of episodic nature with the presence of terminal states, so the agent takes actions until reaching some sort of predefined terminal state. This is the class of problems that is focused on for the following explanations in this chapter. For explanations regarding the non-episodic, *continuing task* case, the reader is referred to [30]. For the episodic case, the cumulated discounted reward from a point t in time until reaching a terminal state at timestep T of an episode can be defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{(T-t-1)} R_T \tag{2.32}$$

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.33}$$

The purpose of the discount factor γ, as the name suggests, is to discount rewards the further they occur in the future. A low value of γ indicates a lower valuation of future rewards compared to immediate rewards in the near future given the current timestep t.

Another useful fact about the cumulated discounted reward G_t , especially regarding the derivation of the Bellman Equation, is that it can be expressed in a recursive fashion:

$$G_t = R_{t+1} + \cdots + \gamma^{(T-t-1)} R_T \tag{2.34}$$

$$= R_{t+1} + \gamma G_{t+1} \tag{2.35}$$

[6] https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

2.4.3 State Value Function

To quantify how good it is for the agent to be in a certain state, the concept of the value function $v_\pi(s)$ was introduced. It measures the expected discounted future reward of a state given that the agent will follow the policy π. In this context, a policy describes the way in which the agent decides on the next action A_{t+1} given the current state S_t it is in. The policy itself can be formulated as a probability distribution $\pi(a|s)$ over all actions $a \in \mathcal{A}$ given state $s \in \mathcal{S}$. It is either deterministic, i.e., given a certain state always one specific action is chosen, or probabilistic, i.e., choosing specific actions with a certain probability. Given this definition of a policy, we can define the value function as

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]. \tag{2.36}$$

Here, the expected value $\mathbb{E}_\pi[G_t|S_t = s]$ is a short form for the following longer term

$$\mathbb{E}_{a\sim\pi(a|s),\ s',r\sim p(s',r|s,a)}\left[G_t|S_t = s\right] \tag{2.37}$$

and indicates the expected value of the cumulated discounted reward from the current state $S_t = s$ at time t on, given the behavior policy π of the agent and the dynamics of the MDP given by $p\left(s', r \mid s, a\right)$.

2.4.4 State Action Function

Similar to the introduced value functions, indicating how good it is to be in a certain state in terms of expected discounted cumulated future rewards, the same can be done for a certain action. Action functions $q_\pi(a|s)$ measure the discounted cumulated future reward given a certain policy $\pi(a|s)$ and state s, that can be expected from timestep t onwards when taking action a and following policy π. It is defined as

$$q_\pi(a, s) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]. \tag{2.38}$$

2.4.5 Bellman Equation

An interesting property of both, state value functions and state action functions, is that they can be expressed in a recursive manner. Assuming a finite MDP and with the help of the property in Equation 2.35, the following can be defined:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] && (2.39)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && (2.40)\\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] && (2.41)\\
&= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right], \quad \text{for all } s \in \mathcal{S} && (2.42)
\end{aligned}$$

The expression in 2.42 is called the *Bellman equation* and is one of the key concepts behind RL theory. It recursively expresses the expected value of the future discounted reward given the agent is currently in $S_t = s$ as sum over all $a \in \mathcal{A}, s' \in \mathcal{S}, r \in \mathcal{R}$ of the reward $R_{t+1} = r$ received in the next timestep and the discounted state value of the next state $S_{t+1} = s'$ weighted by their joint probability represented by $\pi(a \mid s)p\left(s', r \mid s, a\right)$. So the Bellman equation itself could be expressed as an expected value given $S_t = s$. The presented equations only show a part of the complete derivation. For a fully detailed proof of the famous Bellman equation, the reader is referred to the Appendix A.1.2.

2.4.6 Optimal Policy and Bellman Optimality Equation

The goal of the optimization process for RL is to find a policy that, when followed by the agent, on average accumulates high total rewards when interacting with the environment trained on. For example in Tic-Tac-Toe, the agent should learn how to decide on moves that, given the position of all markings already on the board encoded in the state, are favorable for its chances of winning, so on average lead to higher rewards compared to other strategies. The optimal policy π_* can be expressed in terms of the optimal state value or optimal state action function defined as

$$v_*(s) = \max_{\pi} \; v_\pi(s) \quad \forall s \in \mathcal{S} \tag{2.43}$$

and

$$q_*(s,a) = \max_{\pi} \; q_\pi(s,a) \quad \forall s \in \mathcal{S}, \; \forall a \in \mathcal{A}. \tag{2.44}$$

Given a finite MDP and two different policies π, π', one policy is defined as being better than the other, when the expected value of future returns when following this policy is bigger compared to the other policy for all $s \in \mathcal{S}$. As the expected return is defined by $v_\pi(s)$, $\pi \geqslant \pi'$ iff $v_\pi(s) \geqslant v_{\pi'}(s) \;\; \forall s \in \mathcal{S}$.

With this fact and the definition of the optimal state value and action function, the *Bellman optimality equation* can be derived:

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}} q_{\pi_*}(s,a) && (2.45)\\
&= \max_a \mathbb{E}_{\pi_*} \left[G_t \mid S_t = s, A_t = a \right] && (2.46)\\
&= \max_a \mathbb{E}_{\pi_*} \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right] && (2.47)\\
&= \max_a \mathbb{E} \left[R_{t+1} + \gamma v_* \left(S_{t+1} \right) \mid S_t = s, A_t = a \right] && (2.48)\\
&= \max_a \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_* \left(s' \right) \right]. && (2.49)
\end{aligned}$$

The notation $q_{\pi_*}(s,a)$ represents the expected value of cumulated discounted future rewards when taking action $A_t = a$ while being in state $S_t = s$ and assuming that after this interaction, the agent follows the optimal policy π_*. The derived Bellman optimality equation shows that $v_*(s)$ can, similarly to Equation 2.42, be expressed recursively as the expected value over the next reward r and the discounted optimal value of the next state s' given current state s and action a which is maximized over.

While the optimal value function in 2.43 maximizes the state value over the set of all possible policies π, the Bellman optimality equation greedily maximizes over actions a without explicitly considering a particular policy. Since optimizing over all possible policies π is impractical due to the potentially large size of the policy set, the Bellman optimality equation presented in Equation 2.49 provides a more practical optimization target by only maximizing over actions a. This feature makes the Bellman optimality equation an essential tool for finding the optimal policy in complex RL problems.

The same recursive optimality criterion can be formulated for the optimal state action function $q_*(s,a)$:

$$\begin{aligned}
q_*(s,a) &= \mathbb{E}_{\pi_*} \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right] && (2.50)\\
&= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_* \left(S_{t+1}, a' \right) \mid S_t = s, A_t = a \right] && (2.51)\\
&= \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} q_* \left(s', a' \right) \right]. && (2.52)
\end{aligned}$$

Here we do not maximize the whole expression over a, since the action a is included as an argument in the function $q_*(s, a)$ itself . As G_{t+1} stands for the expected future reward from timestep $t + 1$ on, given that the optimal policy π_* is followed, we can replace it by the optimal state action function $q_*(s', a')$, which then needs to be maximized greedily over a' in order to meet the criterion of optimality.

2.4.7 Dynamic Programming

The presented Bellman optimality equation can be used in combination with dynamic programming in order to iteratively find the optimal policy for a RL problem. For this, two different steps are alternated until the algorithm converges to the optimal policy.

The first step, the *Policy evaluation*, sweeps over all states $s \in \mathcal{S}$ and updates the values for $v_\pi(s)$ with the Bellman equation in 2.42 until convergence, i.e., until the real value of all states assuming policy π is reached.

In the second step, *Policy iteration*, the policy π is improved by using the Bellman optimality equation from 2.49 to update π with regard to each of the states $s \in \mathcal{S}$. This way, the optimal policy can be found. However, using this algorithm is only possible under certain conditions which are most of the times met by simpler RL problems. The algorithm assumes a finite MDP and is a *model-based* approach. This means that the dynamics of the environment are known, i.e., $p(s', r|s, a)$ is known and defined. Therefore, the probability of landing in a certain state s' and receiving reward r when taking action a in state s must be known. In most cases, this is a very simplified assumption and not practicable in real world applications. Also, the method represents an example of tabular RL, where each discrete state value $v_\pi(s)$ can be represented as an entry in a big table. As state spaces for many real world scenarios are huge or even infinite, this method is most of the time impractical as well.

2.4.8 Monte Carlo

Compared to Dynamic Programming, Monte Carlo (MC) methods provide a *model-free* RL alternative, enabling to learn a good policy without knowing the exact dynamics of the environment, so with $p(s', r|s, a)$ being unknown. These methods only require the agent to interact with the environment to collect data. The data is represented in the form of trajectories $\tau_i = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T)$ where each trajectory is sampled according to the the underlying policy distribution $\pi(a|s)$ and the environment dynamics distribution $p(s', r|s, a)$. Given this assumption, the following MC estimation can be used as an approximation, e.g., for the state values. The first part of the following is taken from Equation 2.37.

$$v_\pi(s) = \mathbb{E}_{a\sim\pi(a|s),\ s',r\sim p(s',r|s,a)}\left[G_t|S_t = s\right] \tag{2.53}$$

$$\approx \frac{1}{n}\sum_{i=1}^{n}(r^i_{t+1} + \gamma r^i_{t+2} + \gamma^2 r^i_{t+3} + \cdots + \gamma^{T-1} r^i_T) \tag{2.54}$$

$$= \frac{1}{n}\sum_{i=1}^{n} g^t_i \tag{2.55}$$

Here, g^t_i refers to the cumulative discounted reward observed during episode i after state s was visited in timestep t. T is again the terminal timestep. When n is sufficiently large, the MC estimate in 2.55 represents a valid estimate of the true expected value.

The drawback of these methods is that updates to state values or state action value estimates and to the policy π to be learned are only done infrequently after sampling a full episode, since all rewards in an episode need to be known to get g_i for all visited states s.

2.4.9 On-policy vs. Off-policy Algorithms

In general, RL theory distinguishes between two different types of learning paradigms: *On-policy* vs. *off-policy* algorithms.

For on-policy algorithms, the policy followed to sample new actions given a certain state, called the behavioral distribution π_b, is the same as the policy that the model tries to learn, i.e., the policy π_o that is being optimized is identical to the one used to generate new trajectories. In practice, for learning the optimal policy, the agent needs to explore many different states, high as well as low reward regions, to find the best possible one and not get stuck in any local optimum. This policy could be represented by an ϵ-greedy strategy, that chooses the current best action given state s with probability $1 - \epsilon$, but chooses a random action with probability ϵ. For on-policy algorithms, only the employed policy used for sampling trajectories can be learned about.

In contrast, off-policy algorithms update their policy parameters based on experiences generated by a different policy. This can lead to a problem known as the "distributional shift" problem, where the distribution of the experiences we use for learning is different from the distribution of the target policy we want to optimize. This can result in biased estimates of the value function, which can lead to poor performance or even instability of the learning process. When using the MC methods, the expected value is approximated with an MC estimate, which, as explained before, assumes that the samples were taken according to $a \sim \pi_o(a \mid s), \quad s', r \sim p(s', r \mid s, a)$. But in reality, the trajectory would follow $a \sim \pi_b(a \mid s), \quad s', r \sim p(s', r \mid s, a)$, where π_o is the policy that is optimized in the process and π_b the behavioral policy. Therefore, the value approximated by the MC estimate in Equation 2.55 can be written as

$$\mathbb{E}_{a \sim \pi_b(a|s),\ s', r \sim p(s', r|s, a)} \left[G_t | S_t = s \right]. \tag{2.56}$$

If π_b takes some actions very frequently, which the policy currently getting optimized would only rarely choose, the rewards yielded by these actions will be overrepresented in the expected average, while the more relevant ones are underrepresented. This can lead to the mentioned biased estimates caused by this distributional shift. The effect of this shift is particularly strong, the more the two distributions differ. For ϵ-greedy for example, the distributions are still quite similar, but in some other scenarios this might not be the case.

This is why, when using off-policy methods together with MC, importance sampling is needed to correct for this distributional shift. Specifically, the importance sampling ratio is defined as the ratio of the probabilities of taking the actions under the target policy and the behavior policy, given the states encountered during the episode. We can then use this ratio to weight the observed returns, so that they are consistent with the target policy. If certain actions are more likely to occur under the behavioral policy, their influence will be reduced as the term is < 1, and vice versa. This can be derived as follows:

$$\begin{aligned}
v_{\pi_o}(s) &= \mathbb{E}_{\pi_o} \left[G_t \mid S_t = s \right] && (2.57)\\
&= \sum_{\tau_i^t} p_{\pi_o}(\tau_i^t \mid s) g_i^t && (2.58)\\
&= \sum_{\tau_i^t} p_{\pi_o}(\tau_i^t \mid s) \frac{p_{\pi_b}(\tau_i^t \mid s)}{p_{\pi_b}(\tau_i^t \mid s)} g_i^t && (2.59)\\
&= \mathbb{E}_{\pi_b} \left[\frac{p_{\pi_o}(\tau_i^t \mid s)}{p_{\pi_b}(\tau_i^t \mid s)} G_t \middle| S_t = s \right] && (2.60)\\
&= \mathbb{E}_{\pi_b} \left[w(\tau_i^t) G_t \middle| S_t = s \right] && (2.61)\\
&\approx \frac{1}{n} \sum_{i=1}^{n} w(\tau_i^t) g_i^t && (2.62)
\end{aligned}$$

The expected value in the first line can be expressed as a sum over all possible trajectories τ_i^t that could follow timestep t until a terminal state T is reached, given that the agent is in $S_t = s$ at t. The term $p_{\pi_o}(\tau_i^t \mid s)$ represents the probability of trajectory τ_i^t given s under the policy π_o to be optimized. With some transformations, the state value $v_{\pi_o}(s)$ can be expressed as an expected value over trajectories sampled according to the behavioral policy π_b when including the corresponding importance weights $w(\tau_i^t)$. As shown in Equation 2.60, these weights stand for the ratio between the probability of τ_i^t under π_o and π_b, i.e., the probability of the a, r, s triplets following step t until reaching terminal state T. This will rebalance the effects of the distributional shift. As we can see in Equation 2.62, this can yet again be expressed as a MC estimate. The weights $w(\tau_i^t)$ can be computed as follows:

$$w(\tau_i^t) = \frac{p_{\pi_o}(\tau_i^t \mid s)}{p_{\pi_b}(\tau_i^t \mid s)} \tag{2.63}$$

$$= \frac{\pi_o(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t) \dots \pi_o(a_{T-1}|s_{T-1})p(s_T, r_T|s_{T-1}, a_{T-1})}{\pi_b(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t) \dots \pi_b(a_{T-1}|s_{T-1})p(s_T, r_T|s_{T-1}, a_{T-1})} \tag{2.64}$$

$$= \frac{\prod_{k=t}^{T-1} \pi_o(a_k|s_k)p(s_{k+1}, r_{k+1}|s_t, a_t)}{\prod_{k=t}^{T-1} \pi_b(a_k|s_k)p(s_{k+1}, r_{k+1}|s_t, a_t)} \tag{2.65}$$

$$= \prod_{k=t}^{T-1} \frac{\pi_o(a_k|s_k)}{\pi_b(a_k|s_k)} \tag{2.66}$$

One major issue with off-policy MC methods is the high variance of the importance sampled estimates. This variance can be particularly problematic when the ratio of the target policy's and behavior policy's probabilities is large, which can occur when the behavior policy is very different from the target policy. The high variance of the importance sampling estimates can lead to poor estimates of the value function, general instabilities and slow convergence of the learning process.

2.4.10 Temporal-Difference Learning

One of the disadvantageous we have seen for MC approaches in RL in general is that they require the agent to wait until the end of an episode, so until the whole trajectory is known, to update the value function. One approach to mitigate this sample-inefficiency is Temporal-Difference (TD) learning, which is a combination of ideas from the Dynamic Programming and the MC methods. Instead of waiting for the full episode to be sampled, TD learning uses a technique called *bootstrapping* to update the estimates of state value or state action functions in an online fashion with previous estimates. In MC, the estimate of the value of a certain state, i.e, the measure of how good it is for the agent to be in a specific state s, is updated after every new finished trajectory, representing the runnning MC approximation for the expected value. For the value function of a certain state s, the MC updates can be expressed as

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{2.67}$$

or alternatively

$$V(S_t) \leftarrow (1 - \alpha)V(S_t) + \alpha G_t. \tag{2.68}$$

Here, G_t stands for the cumulated future return after visiting state S_t in t in the newest episode. Note that both equations are exactly the same but in the second one it might be even clearer that $(1 - \alpha)$ of the old estimate remains and the proportion α of this old estimate gets replaced by the fresh estimate observed in the current episode. In the book, this method is called *constant-α MC*, since it is not the real average over all the trajectories so far but rather an exponential moving average with a constant update parameter α.

For TD learning on the other hand, this update can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.69}$$

or alternatively

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1})\right]. \tag{2.70}$$

We can see that for the TD case, the agent only takes one step and the estimate for the involved state S_t is immediately updated with the sum of the actual observed reward R_{t+1} and the current bootstrapped estimate for the value of the next state, representing the estimate for the expected future reward from timestep $t+1$ onward. The value for the next state is appropriately discounted with γ as shown in Equation 2.35.

To this day, the TD learning method represents one of the most important and widely-used approaches in RL. As Sutton and Barto [30] state in their book:

> *"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."*[30]

It led to the development of many successful algorithms like the on-policy SARSA or the off-policy Q-learning method. As only the latter will be used in the methods for this work, we will exclusively cover Q-learning in the following section.

2.4.11 Q-Learning

The Q-Learning algorithm is a model-free, off-policy TD control algorithm that was first introduced by Watkins [37]. It directly follows from the theory introduced for TD learning but instead of estimating the value of a certain state given a policy π, it directly approximates q_*, representing the optimal state action function introduced in Equation 2.44 and 2.50. The update rule can be defined analogously to Equation 2.69 above as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{2.71}$$

or alternatively

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right]. \tag{2.72}$$

Similar to the value function, a fraction proportional to α of the current estimate of the value associated with taking action A_t while being in state S_t is updated with the newest estimates observed in the next step of the episode. The current value for $Q(S_t, A_t)$, representing the estimated cumulated discounted future reward assuming that a greedy policy is followed after step t on, is updated with the sum of the actual observed reward R_{t+1} after taking action A_t and the maximum Q-value over all possible actions given that the agent landed in state S_{t+1} as a next state, which is again appropriately discounted with γ. After convergence, the optimal policy can be obtained by selecting the action with the highest Q-value in each state $s \in \mathcal{S}$. As long as the exploration is sufficient, if high as well as low reward areas are explored by the agent, Q-learning will eventually converge to the optimal Q-values and thus the optimal policy. This is guaranteed by the convergence properties of the Bellman optimality equation, which Q-learning is based on. The pseudocode for the tabular case of the algorithm can be found in Figure 2.6.

Another interesting fact about Q-learning is that although it is an off-policy algorithm, in contrast to its presented MC counterparts, no importance sampling is necessary. The reason for this is that during the update only one action is taken but as it is chosen greedily according to the maximum Q-value over all possible actions, it actually represents a sample from the optimal distribution $\pi_o(a|s)$ we want to learn, the greedy policy. This obviously changes when more steps are taken into account as for example in off-policy n-step SARSA [30], which again relies on importance weights.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 2.6: Pseudocode for Q-Learning [30]

2.4.12 Double Q-Learning

One of the main challenges with Q-learning is the potential overestimation of Q-values, which can occur when the action selection and value estimation are both based on the same noisy estimates. This problem is known as the *maximization bias* and is a direct consequence of the maximization operation in the update rule of Q-learning. For a more detailed investigation of this phenomenon, the reader is directed to Chapter 6.7 of the referenced book by Sutton and Barto [30]. This can lead to instabilities during training and suboptimal policies.

To address this issue, several extensions and improvements to Q-learning have been proposed, one of which is Double Q-learning [10]. Double Q-learning is a modification of Q-learning that separates the action selection and value estimation steps by using two separate Q-value functions, the main Q-value and target Q-value function. This approach reduces the overestimation bias by using one of the functions, the main function, to be updated and to choose the next action $s_{t+1} \in S$ using the greedy policy, while utilizing the other function, the target function, to calculate the bootstrapped estimate of the chosen action given s_{t+1}. In each update step, the role of the main and target function is randomly assigned to the two functions, enabling a balanced update procedure. The pseudocode for the Double Q-Learning algorithm can be found in Figure 2.7.

Double Q-learning has been shown to improve the stability and performance of Q-learning in many tasks, especially in environments with high-dimensional state and action spaces.

2.4.13 Function Approximation in RL Methods

All the algorithms and methods introduced in the sections above were based on tabular RL where the current estimates for all state values $v(s)$ or state action values $q(s, a)$ are saved in one big table. For $q(s, a)$ this table would contain all possible combinations of states and actions that could occur in a trajectory. This method works fine for smaller problems with smaller state and action spaces. However, when problems get more complex, the size of this table can become intractably large, making it impractical or even impossible to store and manipulate in memory. In addition, achieving convergence requires the agent to explore the state-action space adequately and perform multiple updates on the Q-estimate for observed state-action pairs. However, this can be challenging in large-scale RL problems where the state and action spaces are vast and the agent may not have enough interactions with the environment to collect sufficient data. As a result, the learning process may be slow or suboptimal, and the agent may fail to generalize to new and unseen situations.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Figure 2.7: Pseudocode for Double Q-Learning [30]

To address the issue with these tabular RL methods, function approximation techniques can be integrated in the learning procedure. Therefore, instead of directly learning each of the values $q(s, a)$ for each possible state and action pair, q will be approximated by a function $q_\theta(s, a)$. This function's parameters are learnable and will be fitted during training to approximate $q(s, a)$ from the given experience of the agent. This task represents a form of regression task from the class of Supervised Learning introduced in Chapter 2.1.1, with q_θ mapping the state and action to a value representing how desirable it is for the agent to take action a in state s with $q_\theta : (s, a) \mapsto \mathbb{R}$.

Analogous to the explanations in the Supervised Learning chapter, there are multiple different choices for models to use as function approximators, such as linear models or decision trees. However, deep neural networks have emerged as the most common choice for approximating the Q-function, especially in complex environments with large state and action spaces. These neural networks have the ability to learn complex, nonlinear relationships between the state and action spaces, allowing the agent to approximate $q(s, a)$ more accurate and to generalize better. By using these function approximators, the agent can efficiently represent and update the Q-function in high-dimensional and continuous spaces, thus overcoming the limitations of tabular RL methods.

Deep Q-Networks (DQN) and the Deadly Triad

Despite the advantages of employing function approximation in RL methods, it is important to note that there are several challenges associated with its use. Not only can it lead to general instabilities during training, but there is also a risk for divergence of estimates. The likelihood of encountering these issues is particularly high when all three elements in the *deadly triad* [30] are present: The use of function approximation, bootstrapping as introduced in TD learning methods and off-policy training causing a distributional shift due to the two different underlying distributions π_o and π_b.

However, research has found methods that mitigate many of the mentioned problems and enable a rather stable training process [23]. In the influential paper by Mnih et al. [21] from DeepMind, the Deep Q-Network (DQN) algorithm was introduced, representing a novel approach that combined deep neural networks with Q-learning to address high-dimensional and complex environments. This paper demonstrated that a single neural network architecture could learn to play multiple Atari games at a human-level or beyond, using only raw

pixel input as the state representation. Two of the changes they made that lead to more stable training process was the introduction of *experience replay* and *infrequent weight updates*.

They introduced the concept of a replay buffer, that stores past experiences from the agent with the environment. Instead of always updating the Q-function with the immediate next state and rewards, for each update experience samples from the replay buffer are randomly chosen to compute gradients and adjust parameters, therefore the name experience replay. New experiences collected by the agent are placed in the buffer to eventually be sampled in a later step. This way, temporal correlations between subsequent states for updates are broken [23]. This also leads to a far higher sample efficiency since one state, action, reward, next step sample is not only used for one update but potentially in multiple updates.

The infrequent updates introduced in their paper helped mitigating divergence problems by using a separate target network, which represents a copy of the main network but is only updated after a certain number of steps.

Double DQN (DDQN)

As we already discussed in Chapter 2.4.12, another problem of Q-learning approaches is overestimation, which can be mitigated by Double Q-Learning [10]. Recently, this method was successfully applied together with function approximation in the form of DQNs [33]. The method is called Double DQN (DDQN) and is an adaption of the standard DQN algorithm to the concept of double Q-Learning. The general weight update rule for Q-learning algorithms can be written as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(Y_t - Q\left(S_t, A_t; \boldsymbol{\theta}_t\right)\right) \nabla_{\boldsymbol{\theta}_t} Q\left(S_t, A_t; \boldsymbol{\theta}_t\right), \tag{2.73}$$

as shown in [33]. Here Y_t^Q stands for the target used in the weight update at step t. This is different for each of the presented algorithms. The following expression represents the target Y_t^Q for the standard Q-learning algorithm with only one network:

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta_t}) \tag{2.74}$$

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
    $y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$
    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

Figure 2.8: Pseudocode for Double DQN (DDQN)[36]

The target for DQN can be written as

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta_t^-}), \tag{2.75}$$

where $\boldsymbol{\theta_t^-}$ stands for the parameters of the target network which are a copy of the main network parameters, which are updated every N^- steps. In standard Double Q-learning, the mentioned target is represented by

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t}); \boldsymbol{\theta_t}'), \tag{2.76}$$

with $\boldsymbol{\theta_t}$ and $\boldsymbol{\theta_t'}$ representing two different sets of parameters for the networks. During training, both networks switch roles randomly, i.e., randomly one of them is used to update its weights and one is used as the target network, leading to a balanced training for both of them. The final target that can be used is the target proposed for the DDQN approach, which is a mixture of the two presented predecessors, Y_t^{DQN} and $Y_t^{DoubleQ}$. It can be written as follows:

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t}); \boldsymbol{\theta_t}-), \tag{2.77}$$

In order to incorporate the benefits of Double Q-Learning while retaining the concept of periodically copying parameters from the main network, the algorithm combines the two approaches. It maintains the idea of copying the main network's parameters every N^- steps, while simultaneously adopting the key insight from Double Q-Learning, which involves separating the maximization step and the actual value retrieval for S_{t+1} and the selected argmax-action a.

3 Related Work and Models

The subsequent chapter initially presents a comprehensive overview of relevant literature, followed by a more in-depth examination of the specific models chosen to explore for the given use case based on the literature review.

3.1 Literature

This section presents some of the most important literature in the field of session-based RS with the focus on approaches that can be applied to sequential data, therefore recommending next items based on the last u most recent interactions. Note that the following literature review is not meant to be exhaustive but rather presents some important examples from each of the categories. For a more detailed collection of methods, the reader is referred to surveys [38], [42], [35] for general, mostly Supervised Learning approaches and this very recent survey [2] for models specifically based on RL.

3.1.1 Supervised Learning Methods

The area of Supervised Learning approaches in the given context of RS is, like many branches of Machine Learning, dominated by deep learning based systems. In the following, three important contributions in this area will be introduced.

One of these contributions is the GRU4Rec model proposed by Hidasi et al. [11] in 2016 for session-based recommendations. In their approach, they leverage the power of RNNs, specifically the GRU cell, to learn long term relationships between inputs of diferent timesteps, and apply this class of models to sequential user-item interactions. Their conducted experiments show promising results on the two included example datasets.

Another important model proposed in 2019 by Tang and Wang [31] is called Caser. In contrast to GRU4Rec, which is based on an RNN architecture, this model uses a convolutional structure. Convolutional layers, frequently found in the realm of computer vision, are one of the main components of a neural network class known as Convolutional Neural Networks (CNNs) [9]. The sequence of items that the user interacted with is embedded into a feature matrix, resembling an image in conputer vision. The authors argue that, while GRU4Rec is based on the sequential processing of the data with high importance of temporal order, Caser on the other hand does not rely on the sequential aspect of user interactions and can learn

local and global patterns in the item embeddings to generate recommendations. In a more detailed comparison, the authors state, that the dataset "*MovieLens has more sequential signals than the other three data sets, thus, the RNN-based GRU4Rec could perform well on MovieLens but can easily get biased on training sets of the other three data sets*" [31]. In their analysis, GRU4Rec performed better compared to the Caser model when applied on the dataset of sequential nature with temporal dependencies, MovieLens[1], than compared to the other datasets which do not have a strong sequential character. On these datasets, Caser was superior.

Another CNN architecture that showed state of the art performance in session-based recommendations is the NextItNet proposed by Yuan et al. [41] in 2019.

As the transformer architecture [34] has proven to be highly effective in various tasks in the area of sequential data representations, first and foremost in NLP, researchers have also explored its potential in the field of session-based RS. One such notable contribution is the SASRec model, proposed by Kang and McAuley [16] in 2018. SASRec is a transformer-based model specifically designed for sequential recommendation tasks. It leverages the power of the self-attention mechanism to capture both short-term and long-term dependencies in user-item interactions. This mechanism enables the model to weigh the importance of different items in a user's interaction history when predicting the next item of interest.

3.1.2 Reinforcement Learning Methods

As we can see from the summary table in the mentioned survey [2], there is a variety of different approaches to apply RL on the general problem of RSs. Many of them are based on methods introduced in Chapter 2.4 like DQN and DDQN. Further details on each of the variations and further algorithms will not be covered in this work, but the references can be found in the survey. This section rather focuses on emphasizing the challenges associated with effectively training RL algorithms for RSs, which in turn underlines the motivation behind the adapted versions of the discussed algorithms.

For RL systems it is crucial that the agent can interact with the environment. With the introduction of DQN or DDQN, it was shown how off-policy algorithms integrating neural networks as function approximators can be efficiently trained. As a reminder, off-policy means that the agent decides on actions according to a behavioral, more explorative policy π_b, while optimizing a different policy π_o, in Q-Learning representing a greedy policy. It is common practice to use an ϵ-greedy policy as π_b, which resembles the optimized policy π_o as it is based on it. So when π_o is updated, π_b changes as well. In the theory part, the risk of distributional shift between π_o and π_b was discussed, that can lead to biased estimates and general training instabilities. It was also presented that these negative effects get stronger when the distributional shift gets stronger, i.e., when π_o and π_b are very different from each other. In most off-policy methods, additional interaction between agent and environment is assumed. This means, while keeping old interactions in the replay buffer to learn from, new episodes are still sampled according to the updated policies [19]. However, in numerous situations, acquiring additional data through interactions with the environment during training is not feasible. This could be due to various reasons. For instance, in autonomous driving tasks, exploration might be unsafe as it could lead to accidents. In the context of RS, this poses a significant challenge as well, since companies would be reluctant to allow real customers to interact with an untrained agent. Particularly, due to the necessary exploration that also needs to cover low reward areas, meaning providing "bad" recommendations, companies risk dissatisfaction of customers. Thus, the only usable data is often customer interactions with an old RS, so an old agent, while trying to train a new advantageous policy. As the behavioral policy employed by the old agent is most likely not optimal, the distributional shift can get rather extreme compared to the ϵ-greedy case, where new samples are generated continuously according to the freshly updated policies. This problem represents

[1] `https://grouplens.org/datasets/movielens/1m/`

an extreme case of off-policy learning and is therefore called *fully off-policy* or *offline* RL [19]. In this given scenario, the agent can in no way interact with the environment and needs to fully rely on a body of offline data to find the best possible policy from it. This poses multiple challenges for the learning process. The explanations in the next paragraphs will all be based on the detailed overview Levine et al. [19] provide in their work, which is why the reader is also directed to this source for more detailed information about offline RL.

First of all, all RL methods require a sufficient exploration of the state and action space to be able to appropriately learn which actions in certain states are desirable with regard to the expected future reward and which should be avoided. In the offline case, this requires the data collected from the previous agent to cover a large part of this state action space. Let us think of a previous RS that was strongly biased towards very popular items and ignored certain others entirely - as there will be no or only very few experiences in the data connected to these unpopular items, no matter how good the algorithm is, the agent would never be able to learn that in a certain state where the old agent proposed a popular item it would have been advantageous to show one of the lower frequency items. It can not be explored from the data as it is not included.

Despite the general instabilities that can arise from a strong distributional shift, this poses a similarly great challenge for evaluation of the algorithms. How do we know that one agent is better than another if only the offline data is given? For example, for the Atari game Breakout that was discussed in Chapter 2.4, the most straight forward way to evaluate an agent trained either online or in an offline fashion, would be to let it interact with the environment, i.e., letting it play the game, and investigating average rewards per episode. A well trained agent would have ideally learned how to win the game, accumulating a high reward each episode on average. When this is not a possibility, as for the autonomous car or the RS, other approximate methods need to be used. The only access to rewards for an offline problem are the rewards present in the dataset. But as the underlying distribution that the sessions included in the data follow might be entirely different from the newly learned one, this would lead to biased estimates of evaluations like the average accumulated reward. As a possibility to mitigate these biases, again, importance sampling can be employed, as introduced in Chapter 2.4.9. However, as explained before, due to high variance this can by itself introduce great instabilities.

As the field of offline RL itself is a complex and vast ongoing area of research, the provided explanations just represent an introduction to the topic. For further details the extensive work by Professor Sergey Levine and his team should be considered [19].

A lot of algorithms in the area of RL for RS use the well known theoretical learning frameworks presented in Chapter 2.4 and try to adapt them to the challenges posed by offline RL as most RS tasks fall into this category. As an example, one of these adaptions is briefly presented in the following. Kumar et al. [18] introduced an algorithm called *Conservative Q-Learning* in 2020. As discussed in Chapter 2.4.12, the general Q-learning approach can suffer from overestimation bias due to the maximization step in the update process, which leads to overly optimistic Q-value estimates. Conservative Q-Learning tries to mitigate this overestimation bias by maintaining a more conservative estimate of the Q-values during the learning process. Instead of simply taking the maximum estimated Q-value in the update step, Q-values that lower-bound the value of a policy π are used. By doing so, the algorithm maintains a more cautious approach in updating the Q-values, reducing the risk of overestimation bias.

3.1.3 Hybrid Methods

In addition to the supervised and RL based methods, there also exist promising hybrid models in the literature that combine both learning paradigms. Note, that the following paragraphs will only give a brief introduction to each of the models, as they will be explained in more detail in the second part of this chapter.

One of the earliest examples for this class of models represents a method proposed by Xin et al. [39] in 2020 called *Self-Supervised Q- learning* (SQN). It represents a neural network based model that has two heads, one supervised and one RL head for accuracy.

Based on this approach the same authors introduced an altered variant of the SQN algorithm called *Supervised Negative Q-learning* (SNQN) [40]. The difference between the two models is that while SQN only uses positive samples to update the Q-values in the Q-head, SNQN introduced a negative sampling procedure to also incorporate negative learning signals into the training process.

In the most recent 2022 paper [29] of a similar set of authors, which also includes a researcher from Googles RL think tank DeepMind, a new approach was proposed that is also based on SQN but adds a multi-objective perspective to it. The model is called *Scalarized Multi-Objective Reinforcement Learning* (SMORL). Rather than just having one RL head for accuracy the architecture features two more additional heads. These heads target diversity and novelty of next item predctions as it was shown that these objectives directly correlate with user satisfaction scores [5], [4]. The authors conduct experiments on two different sequential user-item interaction datasets including successive actions as item views and purchases arranged in standalone user-sessions. For their experiments, they compare among others the GRU4Rec model, the GRU based SQN method and their newest approach, the GRU-SMORL model. Their presented results show, that SMORL performed best in all of the included metrics measuring accuracy, diversity, novelty and repetitiveness, with SQN ranking second and GRU4Rec third. They conclude from these results that adding a RL head for accuracy as in SQN, helps the model to generalize better and that the two additional Q-heads in SMORL further improve this result, especially with regard to the newly identified objectives diversity and novelty. In their abstract the authors summarize that the "*experimental results on two real-world datasets reveal a substantial increase in aggregate diversity, a moderate increase in accuracy, reduced repetitiveness of recommendations, and demonstrate the importance of reinforcing diversity and novelty as complementary objectives*" [29].

Drawing from the encouraging findings presented in the study by Stamenkovic et al. [29] and the comprehensive model comparison they conducted, their proposed approach appears to be a compelling candidate for application in the context of the use case explored in this work. Additionally, the application scenarios explored in their research closely align with the problem faced by IKEA's IF, as the data utilized for training the IF RS comprises user-item or user-image interactions with a temporal ordering, organized into distinct sessions, similar to the data structure in their study. Furthermore, the SMORL model is directly incorporating the additional objectives of diversity and novelty in its optimization procedure, which were previously identified as potential areas of improvement for the IF feed. Altogether, these factors provide a solid foundation for pursuing the procedure outlined in their paper, so applying the SMORL model on the given use case in this work and comparing it to the two other closely related models, SQN and GRU4Rec. Starting with a purely supervised RNN-based approach, followed by two hybrid approaches - one single-objective and one multi-objective - holds the potential to yield valuable insights. Since all three models rely on the same GRU-based backbone, this setup could at the same time facilitate a fairer model comparison, potentially enabling the isolation of the effects of incorporating RL in general and the effects of introducing multi-objectivity into the training process. These insights are particularly important in relation to the research questions set for this work, as they can help assess the impact of RL integration and multi-objective approaches on model performance.

The decision to use GRU as the backbone is further reinforced by the presented findings of Tang and Wang [31], which demonstrated that the GRU4Rec model significantly outperformed Caser in the context of a strongly sequential-based problem, which exactly aligns with the structure of the session based sequential user-item interactions included in the IF data.

3.2 Models

The following section provides a detailed introduction to the three models that will be compared in this work, including the evaluation framework that was used for the study in the original paper [29]. First, the shared backbone of the models is presented, followed by an introduction to GRU4Rec, SQN and SMORL, ending with the explanation of the evaluation metrics.

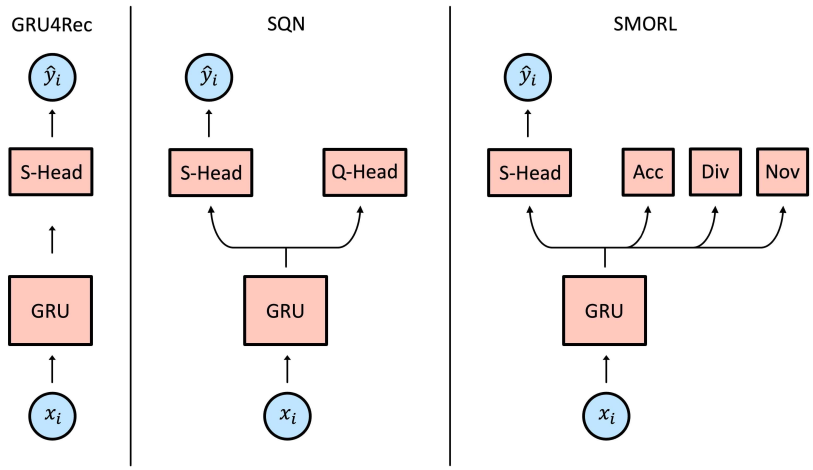


Figure 3.1: Architecture overview of GRU4Rec, SQN and SMORL. x_i is the vector of sample i containing the last u interactions of the customer. $\hat{y}_i$ is the most probable output class, i.e. next item, given the model.

3.2.1 Backbone

Figure 3.1 shows the architecture of the models in a simplified form. We can see that all three models share the same backbone with a GRU at its core. As the authors showed in their experiments in their latest paper [29], this backbone has a modular structure and can be composed of different models. So instead of the GRU shown in the figure, other of the sequential learners for RS introduced in this chapter like transformer based SASRec [16] or the CNN approaches like Caser [31] could be used as well. To narrow the experiment horizon and since the focus of this work is the comparison of the three models as a whole, the same GRU based backbone will be used for all the conducted experiments.

Figure 3.2 shows the first part of the backbone network which is responsible for embedding the user input sequence x_i into vector representations. The example input vector x_i contains the indices of the u latest items the customer interacted with. The state length u, i.e., the number of previous interactions that is provided as an input, is defined during preprocessing, when the replay buffer is created, as shown in Chapter 4. It is the same for each state included in and sampled from the buffer. The embedding layer holds one vector of size m - the previously defined embedding dimension - for each of the item indices included in the training vocabulary. All of the vector representations are learnable parameters that will be trained along with all other model weights. The result of the embedding process is the embedding matrix E_i of size $m \times u$, where each column vector $E_i^1, \ldots, E_i^u$ stands for the vector representation of the item at sequence position $1, \ldots, u$.

The embedding process transforms the input sequence x_i into E_i. The GRU is then unrolled over the sequence, i.e., over $E_i^1, \ldots, E_i^u$, to calculate the final hidden state h_i^u, which can be thought of as a vector type summary of the input sequence. Referring to the theory part in Chapter 2.3, this represents a Sequence-to-Vector Encoder-problem. The described process is shown in Figure 3.3. Starting with the first vector E_i^1, each vector is fed to the GRU cell

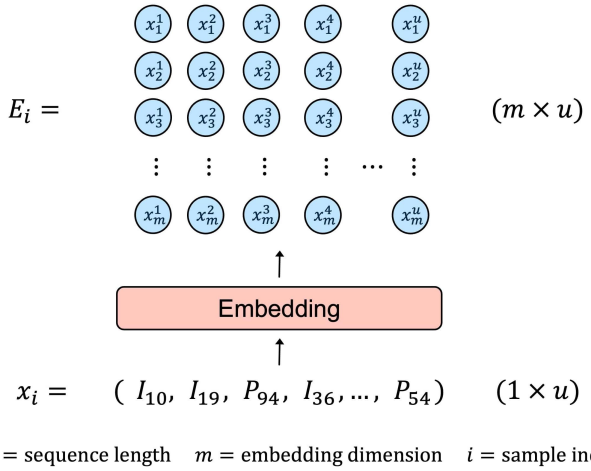


Figure 3.2: Visualization of embedding process as part of the shared backbone. x_i is example click sequence for sample i represented as an index vector of size u by a customer consisting of inspirational image clicks I_j and product clicks P_j. m is embedding dimension, u is length of x_i i.e., the size of the input customer state. E_i represents resulting embedding matrix. Each column vector of E_i is referred to as $E_i^1, \ldots, E_i^u$ in the following.

together with the hidden state from the timestep before, which is then used to compute the next hidden state as shown in Chapter 2.3. h_0 is the hidden state for the first position and is initialized randomly. At each sequence position $1, \ldots, u$, a hidden state is yielded by the model, but in the uni-directional, non-attention based vanilla RNN case, only the last state h_i^u is used and gets passed to the following calculations. It will be referred to as h from now on.

Important to mention is that it is also possible to use multiple layers of stacked GRU cells. The original paper [11] introducing this architecture points out, that in their conducted experiments on the RC15 dataset, additional GRU layers decreased model generalization. Also, in their conducted comparisons of GRU4Rec, SQN and SMORL, only one layer is used [29]. This is why in this work, this structure is maintained as well.

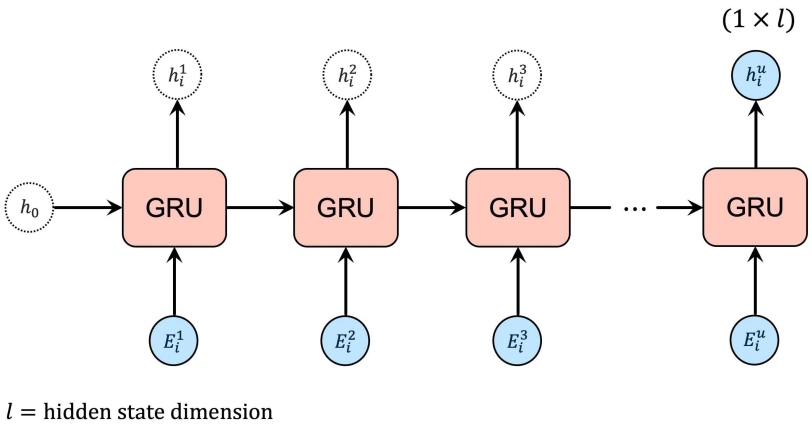


Figure 3.3: Visualization of GRU unrolled over x_i represented by embedding matrix E_i to compute final hidden state h_i^u as summary-vector of x_i. $h_i^1 \ldots, h_i^u$ are hidden states of dimension l yielded by GRU at each sequence position $1, \ldots, u$. Here, only h_i^u as final one is used for following calculations. h_0 is initial hidden state.

3.2.2 GRU4Rec

The GRU4Rec model presented by Hidasi et al. [11] in 2015 is the least complex model of the three in terms of architecture, theory and training procedure. It is entirely based on Supervised Learning and is trained to predict the next item based on the u dimensional previous customer interactions x_i.

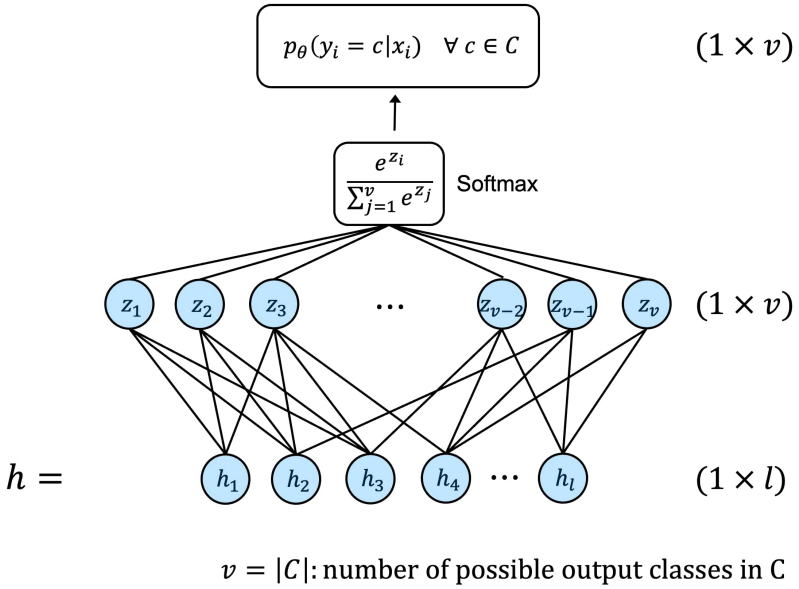


Figure 3.4: Visualization of supervised head in GRU4Rec model. h is last hidden state from backbone GRU of dimension l. z is vector of output logits for each of the $v = |C|$ possible output classes $c \in C$. θ represents model parameters.

As previously mentioned, the model shares the same backbone network as SQN and SMORL, which transforms the input sequence x_i to an l dimensional latent vector h representing a summary of the users most recent interactions. This hidden state is then fed to the supervised head, which is just a feed forward neural network mapping h to the output logits. An overview of this supervised head can be found in Figure 3.4. It specifically shows the smallest version of the head, with only one fully connected layer, encompassing the input layer represented by the last hidden state h passed from the backbone GRU and the output layer. Within this network, the hidden state h is mapped to a vector z containing the output logits for each possible class $c \in C$, where C is the set of all possible next items. The logits are then fed to a Softmax layer, as introduced in Chapter 2.1.2, that transforms them into a discrete probability distribution $p_\theta(y_i = c|x_i)$ over next items $c \in C$. Here, larger logits will be mapped to higher output probabilities for the respective class and vice versa.

This represents the smallest variant of the supervised head without any hidden layers. Additional layers can be added to increase learning capabilities if necessary. In principle every feed forward neural network with the specified input and output dimensions could be used as well. Similar to the number of layers for the GRU, the experiments conducted in the paper [11] revealed that given their use case, adding additional layers to the supervised head did not improve performance.

The loss function used for training the model is the Cross-Entropy loss, introduced in Chapter 2.1.2 in Equation 2.13, which takes as input the real next interacted item y_i and the predicted probability $p_\theta(y_i|x_i)$ of this item. In the original paper [11] the authors looked at alternative loss definitions like Bayesian Personalized Ranking (BPR) or TOP1 ranking loss as well. The results for using these alternative formulations, however, show only slight improvements in accuracy under very specific conditions. This could be the reason why in

the SMORL paper [29], where a GRU4Rec model is investigated as well, the Cross-Entropy loss is used. Therefore, the models trained in this work will follow the same approach.

3.2.3 SQN

As previously explained, the SQN algorithm [39] is the first model of the paper series that includes RL. The main idea of this work is that, by adding an additional head to the base network architecture, in this case a GRU4Rec model, performance and generalization of the RS trained on the sequential session-based user-item interactions can be improved. A high-level overview of this architecture can be found in Figure 3.1 or in more detail in Figure 3.6 which specifically shows the SMORL model. However, the concept of the Q-head is the same. Similar to the supervised head described in the explanations around GRU4Rec, the Q-head takes the encoded hidden state h yielded by the GRU backbone and maps it to a vector of the same dimension as the number of possible next items to show, representing the estimated Q-value for each of these possible actions $a \in \mathcal{A}$ given the state $s \in \mathcal{S}$ that the user is in, represented by the sequence x_i. This mapping can be modeled exactly like for the supervised head by a simple fully-connected layer without hidden layers. Of course, more complex mappings could be used as well. The only difference for the Q-head compared to the supervised head is that no activation function will be applied to the output representing $Q_{head} : h \mapsto \mathbb{R}^{|\mathcal{A}|}$.

The training procedure for the supervised loss does not change compared to GRU4Rec. The new Q-head follows the Double Q-Learning method, not the DDQN version, but rather the basic Double Q-Learning algorithm, with the target represented in Equation 2.76. This means that two different copies of the network will be trained simultaneously, and in an alternating random fashion, one is used as the target network and the other as the main network getting updated in this step. The pseudocode for the full algorithm can be found in Figure 3.5. As loss function, the MSE-loss is used.

Algorithm 1 Training procedure of SQN

Input: user-item interaction sequence set $\mathcal{X}$, recommendation model G, reinforcement head Q, supervised head
Output: all parameters in the learning space Θ

Initialize all trainable parameters
Create G' and Q' as copies of G and Q, respectively
repeat
  Draw a mini-batch of $(x_{1:t}, a_t)$ from $\mathcal{X}$, set rewards r
  $\mathbf{s}_t = G(x_{1:t})$, $\mathbf{s}'_t = G'(x_{1:t})$
  $\mathbf{s}_{t+1} = G(x_{1:t+1})$, $\mathbf{s}'_{t+1} = G'(x_{1:t+1})$
  Generate random variable $z \in (0, 1)$ uniformly
  if $z \leq 0.5$ **then**
    $a^* = \text{argmax}_a \, Q(\mathbf{s}_{t+1}, a)$
    $L_q = (r + \gamma Q'(\mathbf{s}'_{t+1}, a^*) - Q(\mathbf{s}_t, a_t))^2$
    Calculate L_s and L_{SQN} according to Eq.(5) and Eq.(8)
    Perform updates by $\nabla_\Theta L_{SQN}$
  else
    $a^* = \text{argmax}_a \, Q'(\mathbf{s}'_{t+1}, a)$
    $L_q = (r + \gamma Q(\mathbf{s}_{t+1}, a^*) - Q'(\mathbf{s}'_t, a_t))^2$
    Calculate L_s and L_{SQN} according to Eq.(5) and Eq.(8)
    Perform updates by $\nabla_\Theta L_{SQN}$
  end if
until converge
return all parameters in Θ

Figure 3.5: Pseudocode for SQN algorithm [39]

According to the authors, the idea, why the additional head can help to improve results is that it "*acts as a regularizer to fine-tune the recommendation model G according to our reward settings*" [39]. Therefore, there is no direct influence of the RL head on the recommendations, since they are yielded by the supervised head. However, for updating the parameters in

the GRU backbone, which they refer to as the recommendation model G, the gradient from this head, so essentially from the Q-loss, will be flowing into the backbone, i.e., the GRU model and the embedding vector representations of the items. This way, the gradient used to update the parameters in the backbone will be a mixture of the gradient backpropagated from the supervised Cross-Entropy loss and the Q-loss based on the current data batch. The reward is user defined and should be designed in a way that particularly desirable actions get assigned a higher positive reward, while less or non-desirable actions get assigned a less positive or negative reward. For their experiments, only two types of actions are included in the data, item views and purchases, and in their experiments they found that $r = 0.2$ for views and $r = 1$ for purchases worked best [39]. Reward design in general is a vast subject by itself, which is why the reader will be referred to Chapter 17.4 in [30] for more details on this topic.

Intuition for Q-Learning in the context of RS

The following paragraph provides a general intuition for what Q-Learning in the context of RS means, what happens during updates and what is learned. Keep in mind that this explanation is not specifically focused on the Q-heads in SQN, but rather pertains to the general application of Q-Learning for RS. However, since Q-heads are trained using this method, the explanation is still relevant to them as well. For each sample of user sequence x_i, the Q-loss is computed for the Q-value corresponding to the real next interaction by the customer and the associated reward. So the next item the user interacted with, denoted as y_i, is used as action a for $Q(s = x_i, a)$. It is essential to understand that this action must be viewed from the perspective of the RS agent rather than the customer. The action represents the next item shown to the user by the RS - otherwise they could not have clicked it - meaning it is the RS agent's action. The reward measures how desirable the user's subsequent reaction or behavior was, given the pair of customer state $s = x_i$ and RS agent's action a. For example, the user might continue browsing items and eventually make a purchase, a desirable outcome for the company, so the state-action pairs that lead to this trajectory should be reinforced by the flow of positive rewards into their Q-estimates. On the other hand, if the user decides to end the session soon after the RS took action a and recommended the corresponding item, this would be an undesirable outcome for the company. As a result, there would be little or no positive reward flow in the corresponding $Q(s, a)$ values, since an undesirable terminal state is near. It is important to note in this context that for a terminal state, $Q(s_{\text{terminal}}, a) = 0$ for all actions a in the action set $\mathcal{A}$. The user essentially acts as the environment of the MDP in this context, reacting to the actions taken by the agent, i.e., recommending a certain item. The observed next state s' provides insights into how the environment has changed as a result of the previous action a. It contains information about the updated user state, which, for an RNN-based sequential model like SQN, is represented by the modified interaction history of the user, i.e., appending the most recently clicked item to the stack of previously interacted items. The descriptions were based on one singe example. However, the updates to the weights of the Q-function are of course done on a whole batch of customer samples with many different learning signals, which lead to an average gradient-direction for the update. Its application for the weight updates will decrease the Q-losses inside this specific batch - on average.

3.2.4 SMORL

The SMORL model [29] is closely related to SQN but features two additional Q-heads, transforming the RL part of the model to a multi-objective approach. As mentioned before, the two new Q-heads serve the purpose of integrating reward feedback of recommendation diversity and novelty into the learning procedure. The authors argue that these objectives are positively correlated with ease of use, perceived usefulness and general satisfaction with the RS [5], [4], [14].

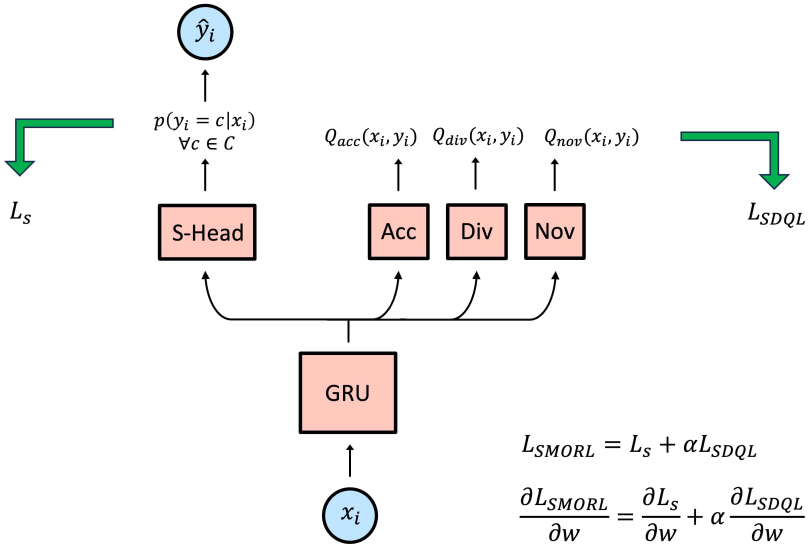


Figure 3.6: Overview of SMORL architecture. States s and actions a for the different Q-values are replaced by the values used in the actual model. s is defined as the prior interaction sequence x_i of a customer and for a the next interacted item is utilized, which is represented by y_i. C is the set of all possible next items.

An overview of the model is provided in Figure 3.6. The full pseudocode of the model can be found in Figure 3.7. Again, the theoretical framework used to train the Q-heads is basic Double Q-Learning with a neural network as function approximator and a slight modification due to the multi-objective nature of the model. The Q-head on the left is the *Accuracy Head*, which is similar to the single head used in SQN. According to the authors, this head helps to reinforce accuracy and, *"from the definition of the reward, the model is rewarded when it matches the next clicked item in the sequence"* [29]. The heads to the right on the other hand should help to reinforce the two new objectives novelty and diversity. Each of them undergoes the same training process. The only difference is the reward signal they receive. Each of these rewards is meant to target the specific objective, reinforcing desirable and discouraging undesirable state-action pairs.

The computation of the loss tied to the RL heads introduces two new hyperparameters, the objective weighting vector w and α. The following equations demonstrate, how the loss for the whole model, L_{SMORL}, it computed from the supervised Cross-Entropy loss L_s and the loss for the RL-regularizer L_{SDQN}:

$$L_{SMORL} = L_s + \alpha L_{SDQN} \tag{3.1}$$
$$L_{SDQN} = f(L_{acc}, L_{div}, L_{nov}) \tag{3.2}$$
$$= (L_{acc}, L_{div}, L_{nov}) \cdot w \tag{3.3}$$

After each Q-loss with regard to the given head is calculated, they are aggregated into one unified loss signal L_{SDQL}, where SDQL stands for *Scalarized Deep Q-Learning* [29]. As illustrated in Equation 3.2, this aggregation is achieved by inputting them into a predefined function f which maps them to the RL loss signal L_{SDQL} [29]. The authors propose a linear mapping with an according weight vector $w \in \mathbb{R}^3$ that represents the weighting between the three different objectives, as shown in Equation 3.3. However, they also point out that f is a design choice and can also be defined as a non-linear mapping.

The second introduced hyperparameter, denoted by α, regulates the balance between the loss signal of the supervised head and L_{SDQL}, the one yielded by the RL part of the model.

This weighting essentially influences the gradient intensity of the supervised part on the one hand and the Q-learning driven RL part on the other hand. By elevating the value for α, the loss signal from the RL heads is boosted, in turn boosting the gradient that is flowing into the bottom layers of the model. Similar to the explanations for the SQN model, the gradient signal reaching the backbone layers during each batch update will be a mixture of the gradient flow from the supervised part and from the RL part. With a higher α value, this mixture leans more towards the RL heads, allowing a greater influence on the parameters within the GRU-cell and the embedding vectors, and vice versa.

Each of the individual Q-losses L_{acc}, L_{div} and L_{nov}, used in Equation 3.3, is computed according to an adapted version of Double Q-learning introduced in Chapter 2.4.13 as

$$L_{obj} = (Y_t^{obj} - Q_{obj}(S_t, A_t; \boldsymbol{\theta_t}))^2 \quad \forall\ obj \in \{acc, div, nov\} \tag{3.4}$$

and

$$Y_t^{obj} = R_{t+1}^{obj} + \gamma Q_{obj}(S_{t+1}, a^*; \boldsymbol{\theta_t}'). \tag{3.5}$$

with

$$a^* = \underset{a}{\text{argmax}} \begin{pmatrix} Q_{acc}(S_{t+1}, a; \boldsymbol{\theta_t}) \\ Q_{div}(S_{t+1}, a; \boldsymbol{\theta_t}) \\ Q_{nov}(S_{t+1}, a; \boldsymbol{\theta_t}) \end{pmatrix}^T \cdot w. \tag{3.6}$$

For each of the Q-heads, one loss signal L_{obj} is computed based on the current state S_t, which is represented by the customer sequence x_i, the action A_t, which is the true next item y_i interacted with in the next timestep, and the next state S_{t+1}, which refers to the next state the customer lands in after clicking the item in y_i. As explained in the paper, in the case of session based recommendations this is the updated customer interaction sequence. R_{t+1}^{obj} is the reward connected to the tuple S_t and A_t, which is included in the replay buffer. θ_t and θ_t' are the two separate sets of model weights that are optimized in parallel. The only difference to the standard Double Q-Learning algorithm from Equation 2.76 in Chapter 2.4.13, is the choice of the next action a^* used for bootstrapping in the target. In the presented multi-objective case, for the choice of a^*, the Q-value of all three objectives is incorporated into the decision, each of them weighted according to w. This is shown in Equation 3.6. Based on this weighted sum of Q-values for each possible next action A_{t+1}, the one with the highest value is chosen as the best next action to take with regard to all three objectives.

All of the explained calculations, with slightly different notation, can also be found in the pseudocode from the original paper [29], which is presented in Figure 3.7.

Accuracy Reward

The reward for the accuracy head in the case of SQN is an offline reward that will be assigned to each of the s, a, s' triplets in the replay buffer. This reward is a design choice and as mentioned before can be varied for different scenarios (e.g. buy vs. view) [39]. In the SMORL paper, they provide a more general formulation that just assigns 1 online to each of the state-action pairs visited during training, since the action in this case represents the true customer interaction which shall be reinforced. This way, the method "*can be easily extrapolated from e-commerce to other relevant areas of RS*" [29]. While recommending this reward procedure for the accuracy head, based on their provided original implementation[2] they used for the paper, it is clear that they actually use the same reward setting as in SQN and not the one described earlier. A code snippet illustrating the utilized approach is shown in Figure A.11 in the Appendix.

[2]Original SMORL implementation can be downloaded from `https://drive.google.com/file/d/1lVeKlajOkZ4n9Rl2VmJvYR9i1aXWkR2j/view`

Algorithm 1: Training Procedure of SMORL

Input : user-item interaction sequence set $\mathcal{X}$,
recommendation model G,
SMORL head $\mathbf{Q}$, supervised head S,
predefined parameters α and $\mathbf{w}$
Output : all parameters in the learning space Θ

Initialize all trainable parameters
Create G', $\mathbf{Q}'$, as copies of G and $\mathbf{Q}$, respectively
repeat
  Draw a mini-batch of $(\mathbf{x}_{1:t}, a_t)$ from $\mathcal{X}$
  $\mathbf{s}_t = G(\mathbf{x}_{1:t}), \mathbf{s}'_t = G'(\mathbf{x}_{1:t})$
  $\mathbf{s}_{t+1} = G(\mathbf{x}_{2:t+1}), \mathbf{s}'_{t+1} = G'(\mathbf{x}_{2:t+1})$
  Generate random variable $z \in (0, 1)$ uniformly
  if $z < 0.5$ **then**
    $a^* = \text{argmax}_a[\mathbf{Q}(\mathbf{s}_{t+1}, a) \cdot \mathbf{w}]$
    pred = argmax $\mathrm{S}(\mathbf{s}_t)$
    Set reward $\mathbf{r}_t = \text{stack}(r_{\text{acc}}, r_{\text{div}}, r_{\text{nov}})$
    $L_{\text{SDQL}} = \mathbf{w}^\top(\mathbf{r}_t + \gamma \mathbf{Q}'(\mathbf{s}'_{t+1}, a^*) - \mathbf{Q}(\mathbf{s}_t, a_t))^2$
    Calculate L_s
    $L_{\text{SMORL}} = L_s + \alpha L_{\text{SDQL}}$
    Perform updates by $\nabla_\Theta L_{\text{SMORL}}$
  else
    $a^* = \text{argmax}_a[\mathbf{Q}'(\mathbf{s}'_{t+1}, a) \cdot \mathbf{w}]$
    pred = argmax $\mathrm{S}(\mathbf{s}_t)$
    Set reward $\mathbf{r}_t = \text{stack}(r_{\text{acc}}, r_{\text{div}}, r_{\text{nov}})$
    $L_{\text{SDQL}} = (\mathbf{w}^\top(\mathbf{r}_t + \gamma \mathbf{Q}(\mathbf{s}_{t+1}, a^*) - \mathbf{Q}'(\mathbf{s}'_t, a_t))^2$
    Calculate L_s
    $L_{\text{SMORL}} = L_s + \alpha L_{\text{SDQL}}$
    Perform updates by $\nabla_\Theta L_{\text{SMORL}}$
  end
until *converge*;
Return all parameters in Θ

Figure 3.7: Pseudocode for SMORL algorithm [39]

Diversity Reward

The authors determine the diversity reward in an online fashion, i.e., calculated after the actual recommendations are yielded for one batch during training, depending on the similarity of the top prediction of the current RS by the supervised head, and the last, most recently interacted item in the user history. To measure how similar two items are, the authors make use of an approach especially known in the context of NLP. They retrieve a similarity measure of two items by calculating the cosine similarity [32] of their embedding vectors. The idea is that the embedding layer is trained in a way that similar items (= words in NLP) get assigned closer vectors that point in similar directions in the vector space. A higher cosine similarity means that the angle between the two embedding vectors is small, indicating that they are pointing in similar directions and thus are more similar. Conversely, a lower cosine similarity means that the angle between the two embedding vectors is large, indicating that they are pointing in different directions and thus are less similar. Figure 3.8 shows an illustration of the cosine similarity using fictional two dimensional embeddings of three IKEA IF images as examples. On the left, it is observable that the vectors of the two items or images point in similar direction, which is indicated by the angle θ which is $< 90^\circ$. All angles smaller than 90° are mapped to values in the interval $(0, 1]$, with 1 for the cosine similarity of the same vector with itself. For exactly 90° it is 0 and for all angles over that, the value falls in the interval $[-1, 0)$, with -1 indicating vectors of opposite directions.

Important to note is that the embeddings of items are not retrieved from the embedding layer that is trained together with the model. First of all this layer is learnable as well and will change during training, starting with randomly initialized item vector representations and gradually learning them from experience. Thus, this would lead to instabilities in reward assignment as the rewards in early epoch would be random, leading to wrong and

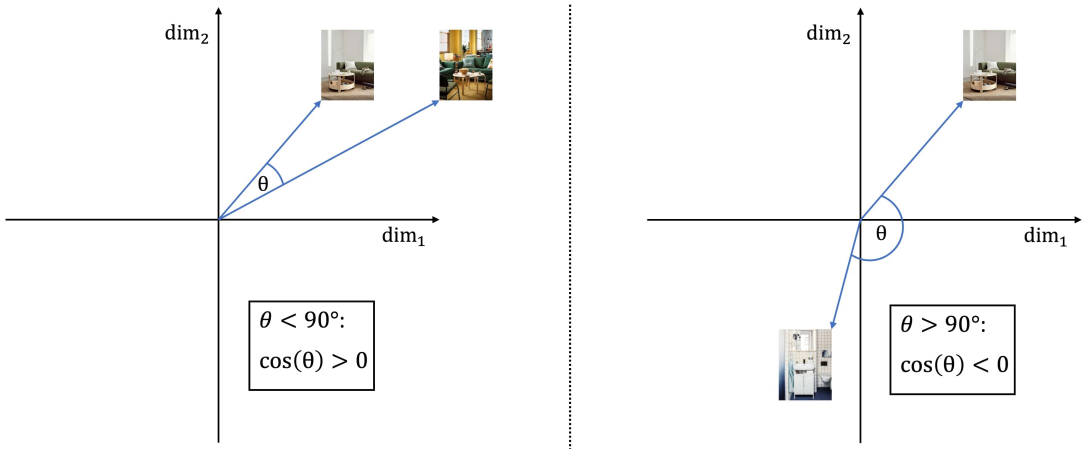


Figure 3.8: Illustration of cosine similarity for IKEA IF images as example items. The arrows represent exemplary vectors of the items given the two dimensional space. Left: Two item-vectors with similar direction. Right: Two item-vectors with more divergent directions. θ is the angle between item-vectors.

random diversity rewards and thus biased diversity learning signals. On the other hand, these rewards should be stable across multiple model runs as well, to make training comparable. This is why the authors propose to use pretrained, frozen embeddings that will be fixed for all model runs [29]. To train them, they suggest to run a GRU4Rec model and extract the embedding layer afterwards and load it in each new model run as the basis for the diversity rewards.

To compute the actual rewards from the cosine similarity, the following formula is applied

$$r_{\text{div}}(s,p) = 1 - \cos(c,p) \quad \text{with} \quad c = \text{last interacted item} \tag{3.7}$$

$$= 1 - \frac{e_c^T e_p}{\|e_c\|\|e_p\|}, \tag{3.8}$$

where p stands for the top prediction by the supervised head given state s and e_c and e_p represent the according embedding vectors.

Important to note is that $r_{\text{div}} \in [0,2]$. So two vectors of the same direction indicating the same item will be assigned a reward of 0. For two items with embeddings pointing in opposite directions, the cosine similarity will be -1, yielding a reward of 2.

It may initially seem counterintuitive to reward a RS when the last interacted item c and the top recommendation p are substantially different in terms of their cosine similarity. However, the authors present the RL component as a regularizer, preventing the supervised head from consistently recommending items that are too similar [29]. This approach will, according to them, encourage diversity in the recommendations generated by the system.

Novelty Reward

In sales, it is a common phenomenon that only a small proportion of highly popular items is sold in large quantities, while the majority of items, typically niche or less popular products, are sold in smaller quantities [3]. This distribution of sales often resembles a long-tailed power law distribution, hence the name "long tail". For a practical example of such a distribution, the reader is referred to the frequency distribution of IKEA's IF images in Figure 5.2.

For the novelty reward, the authors argue that "*less popular items are more likely to be novel and lead to a more balanced distribution of item popularity*" [29]. This way they propose a repre-

sentative binary measure for novelty rewards:

$$r_{\text{nov}}(p) = \begin{cases} 0 & p \text{ in top } x\% \text{ of most popular items} \\ 1 & \text{otherwise} \end{cases} \tag{3.9}$$

Here, p is again the top prediction from the supervised head.

What makes RL attractive?

Before delving into the evaluation section, it is worth highlighting one aspect of RL that makes it particularly appealing for incorporating non-standard objectives such as diversity and novelty, as emphasized by the authors [29]. In Supervised Learning, as seen earlier in this work, learning the parameters of any given function f_θ requires an appropriate loss function that captures the objective. Given that the field of parameter optimization in neural networks predominantly relies on gradient updates and some form of gradient descent, the primary prerequisite for such a loss function is differentiability. If no gradient can be computed, this learning procedure is not suitable at all.

For the application of RL on the other hand, it only requires to provide a well defined scalar reward signal to the model, enabling the integration of objectives that are not expressible as differentiable functions or for which an appropriate function might not even be known. This characteristic makes RL methods attractive for goals like diversity or novelty, whose loss definitions might not be differentiable or hard to formulate in general.

3.3 Evaluation Framework

Particularly when dealing with relatively new and non-standard objectives like novelty and diversity in conjunction with the offline nature of the RS problem, it is crucial to establish a robust evaluation framework that enables monitoring of changes across the various dimensions. The evaluation procedure used in the original SMORL-paper [29] includes metrics for accuracy, diversity, novelty and repetitiveness.

3.3.1 Accuracy metrics

To quantify the accuracy of the RS with regard to the collected sequential customer interactions contained in the dataset, two different metrics are used, the Hit Ratio (HR) and the Normalized Discounted Cumulative Gain (NDCG).

Hit Ratio

The hit ratio HR@k measures the accuracy of the model with regard to its top-k predictions. It indicates the proportion of times that the real next item is included in the top-k recommendations of the model. The metric is computed as follows

$$\text{HR@k} = \frac{H_k}{N}. \tag{3.10}$$

H_k stands for the hits in the top-k predictions, so the number of times the real next item is included in the k most probable items predicted by the model. N is just the total number of samples. HR@1 represents the usual definition of accuracy.

NDCG

The NDCG@k is closely related to the hit ratio but additionally takes into account the ranking of the true item within model predictions. It assigns higher values when the true label has a high ranking and lower values if the model predicts the real item with lower ranking. This

metric captures not only the accuracy of the recommendations but also the importance of their relative positions in the predicted rankings. It can be computed as follows [15]:

$$\text{NDCG@k} = \frac{\log_2(\text{rank}+1)^{-1}}{N} \tag{3.11}$$

For the best ranking, rank $= 1$, the numerator of the fraction is 1, yielding the same value as the HR@k metric. If the ranking of the true next label is greater than 1, the logarithm will be > 1, leading to a numerator smaller than 1. Assuming two recommendation models had the same hit ratios, the one with the higher NDCG value should be preferred, as it indicates that on average the true next item is higher ranked, more closely mirroring the customer behaviour included in the interactions it was trained on.

One notable advantage of the hit ratio metric over NDCG is its ease of interpretability. The hit ratio represents the average number of successful predictions within the top-k recommendations, providing a straightforward measure of performance. In contrast, NDCG does not offer the same level of interpretability.

3.3.2 Diversity

For diversity, the authors suggest measuring Coverage (CV) by considering the proportion of all possible items that are included in the top-k predictions made for samples in the validation or test dataset. It indicates the proportion of total items that are shown by the model given the customer interactions processed during evaluation. It is computed as follows, with I_{total} representing the total number of possible items:

$$\text{CV}_{\text{div}}\text{@k} = \frac{\text{\# of unique items covered in top-k preds}}{I_{total}} \tag{3.12}$$

It is important to mention that this measure is one way of evaluating diversity, by checking which ratio of items was covered during the evaluation procedure. However, it does not provide insights for example in how often certain items were predicted. One could think of a situation in which a certain item gets recommended for each customer state included in the evaluation set and all other items would be recommended only once during the whole evaluation. The coverage would be 1 even though the diversity of the recommendations is obviously not desirable.

Another aspect to consider when analyzing or comparing CV_{div} is its dependence on the size of the dataset used for calculation. If more samples are included in the evaluation procedure, the model has greater opportunities to predict new items that have not been covered previously, which ultimately increases coverage. Thus, when comparing CV_{div} between different model runs, it is crucial to ensure that the calculations are based on the same dataset or at least on datasets of similar length.

3.3.3 Novelty

The metric for novelty is closely related to CV_{div}. However, instead of calculating item coverage over all possible items, for novelty only the unpopular long-tail items are included. Based on the set of less popular items, the novelty measure is computed as follows, with I_{unpop} representing the set of less popular long-tail items:

$$\text{CV}_{\text{nov}}\text{@k} = \frac{\text{\# of unique items} \in I_{unpop} \text{ in top-k preds}}{I_{unpop}} \tag{3.13}$$

3.3.4 Repetitiveness

The authors also introduce a metric for repetitiveness in their work [29]. R@k measures the average number of repetitions in the recommendations in one of the sessions included in the

validation dataset. They define it as

$$\text{R@}k = \frac{1}{S}\sum_{i=1}^{S} \text{\# repetitions in top-k preds of session } i, \tag{3.14}$$

with S as the number of sessions.

4 Data

The following chapter provides an overview of the data that is underlying the experiments conducted in this work. In the first section, the raw clickstream data will be described and briefly analyzed. The second section presents the preprocessing steps that are taken, followed by the data split. The clean and splitted dataset will be the basis for the creation of all the replay buffers for the experiments, which will be discussed in more detail in Chapter 5.

4.1 Raw Clickstream Data

The data underlying all the experiments in this work is clickstream data from the IKEA website that was collected during a period of 41 days from the 5th December 2022 until the 15th January 2023.

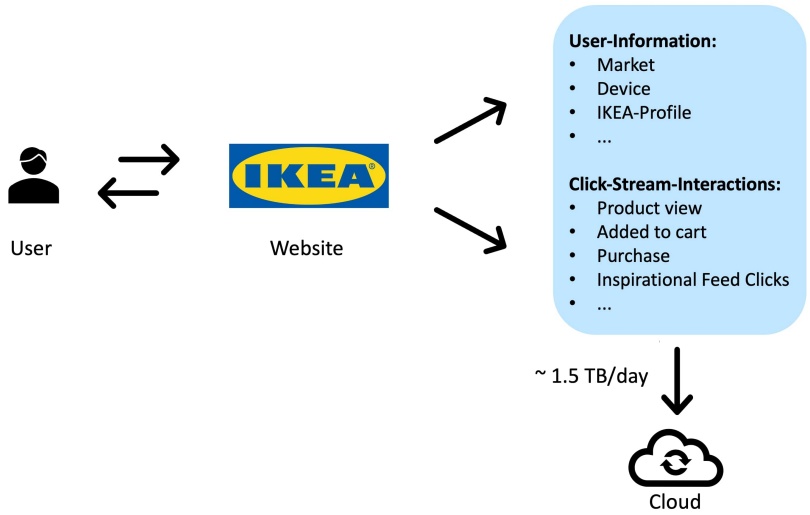


Figure 4.1: Overview of data collection process for clickstream data at IKEA

In general, clickstream data is a type of data that is generated as users navigate through a website or application. It is a sequential record of the pages that a user visits, including the

individual timestamp and duration of their visit on each subpage. Clickstream data can also include information such as the user's geographic location, device type, browser type, and the actions they took on each page, such as clicks, scrolls, and form submissions.

An overview of the process at IKEA for collecting this type of data is illustrated in Figure 4.1. Around 1.5 terabytes of data are acquired and saved to the cloud every day. In particular, for IKEA, this data on the one hand contains information about the user like the market of the customer, i.e. which country they are coming from, or the IKEA profile that is connected to the user. On the other hand, different kinds of clickstream interactions are gathered, such as product views, purchases, adding an item to favorites or the IF clicks, which are central to the research objective investigated in this work.

Table 4.1: Example extract from the raw IKEA clickstream dataset

	start_time_ms	visitor_id	market	timestamp_ms	action	item_id
0	1672337131000	134024...	ae	1672599101686	view_item	50298418
1	1672337131000	134024...	ae	1672599131269	view_item	50178411
2	1672337131000	134024...	ae	1672599148518	view_item	50178411
3	1672337131000	134024...	ae	1672649406160	view_item	50178411
4	1672337131000	134024...	ae	1672649424488	view_item	30488966
5	1672337131000	134024...	ae	1672649493390	click_insp	ee7f1af0-b8a...
6	1672337131000	134024...	ae	1672649509344	click_insp	879be313-4ab...
7	1672337131000	134024...	ae	1672649522467	click_insp	5afa5cdb-0f3...
8	1672337131000	134024...	ae	1672649537856	click_insp	b4dcec9d-d5f...
9	1672337131000	134024...	ae	1672649958347	click_insp	5963d9cc-99e...

As mentioned above, the data used in this work is a subset of the body of described user-clickstream records collected over a period of 41 days. The raw data before preprocessing is queried from the IKEA database and filtered in the process to only include sessions of users that actually contain any clicks on inspirational images. This entire body of records contains a total of 73,760,317 customer interactions with the IKEA website. A small excerpt from the raw dataset is included in Table 4.1. Here, columns that are not important for the further process were excluded for better clarity.

In the following, the data is analyzed and visualized briefly to offer the reader a better overview over its main characteristics. However, it must be noted that for producing these metrics and plots, only a smaller subset of the data containing around 10 million samples is used, which should still be a reasonable representative for the full body of records.

The first three columns of the dataframe represent the start time of a certain session, the corresponding (masked) visitor ID and the geographical market which the user belongs to. These three characteristics clearly identify a unique user-session. This can be thought of as the clickstream of one specific user over a certain period of time. There are many factors influencing what exactly is considered as one coherent session and this is an IKEA internal characteristic. In over half of the cases this will just be clicks collected over one single day as we can see from the left plot in Figure 4.2. But in other examples, one big session can contain multiple smaller inter-sessions[1], which will occur in relatively short periods of time but do not necessarily need to be recorded on the same day. As observable in the plot, some sessions last for more than a few days. The visual cutoff is at ten days, but even sessions lasting for the entire data horizon of 41 days were occasionally recorded. The third quartile is around the seven day mark, so 75% of the sessions refer to collected customer interactions during a period of up to one week.

The plot on the right side of Figure 4.2 shows the distribution of the number of inter-sessions in in each of the roughly 400 thousand individual sessions. It shows that sessions

[1] A smaller session within a larger session, e.g. browsing on one day and then continue browsing one day later.

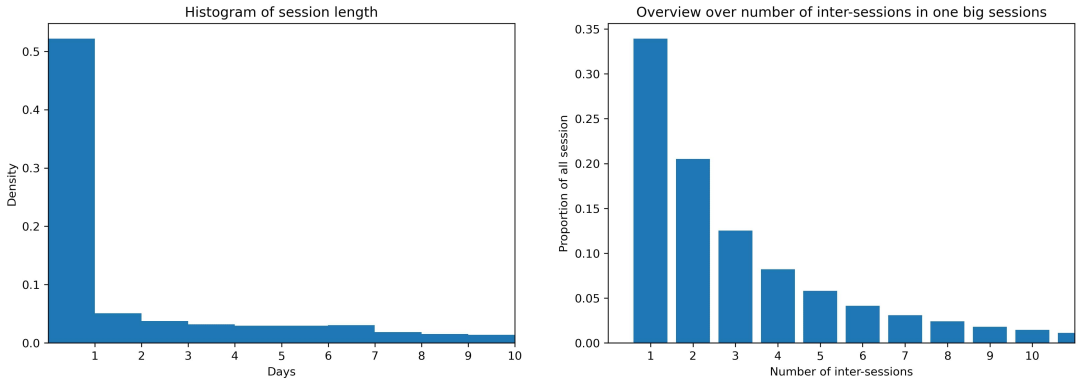


Figure 4.2: Left: Histogram of big session lengths measured in days, visual cutoff at ten days. Right: Overview over number of inter-session per big-session, visual cutoff at ten inter-sessions.

with higher number of inter-sessions are less likely and more than 50% of the sessions only contain one or two of them (probably over a short amount of time). As mentioned before, the definition and properties of how to record sessions are IKEA internal specifications and for the purpose of this work not of great relevance. It is just important to keep in mind that one session is not limited to browsing the IKEA website during one day but can also include multiple consecutive visits on different days in short time intervals of a few hours or days.

The rest of the table columns refer to the individual recorded action at the exact timestamp in column `timestamp_ms`. The `action` column indicates which action type the interaction represents. This can be the view of a product or a click on an IF image for example. Besides these two classes of actions there are purchases, adding an item to the cart or the personal wishlist. The most frequent actions in the dataset are product clicks which account for over 75% of the total interactions. Adding items to the cart and the clicks on the IF images both make up around 10%. An additional observation to highlight is that more than fifty percent of all sessions comprise only a single instance of an IF image click. This is clearly demonstrated in Figure 4.3, which presents the distribution of IF image clicks per session across the entirety of the dataset. We can see that over 90% of all sessions contain five or less IF interactions and the percentage of sessions featuring a greater number of interactions decreases as the number of clicks increases. In essence, there is an inverse correlation between the frequency of sessions and the number of IF clicks they contain. This is important to remember when defining the method later on.

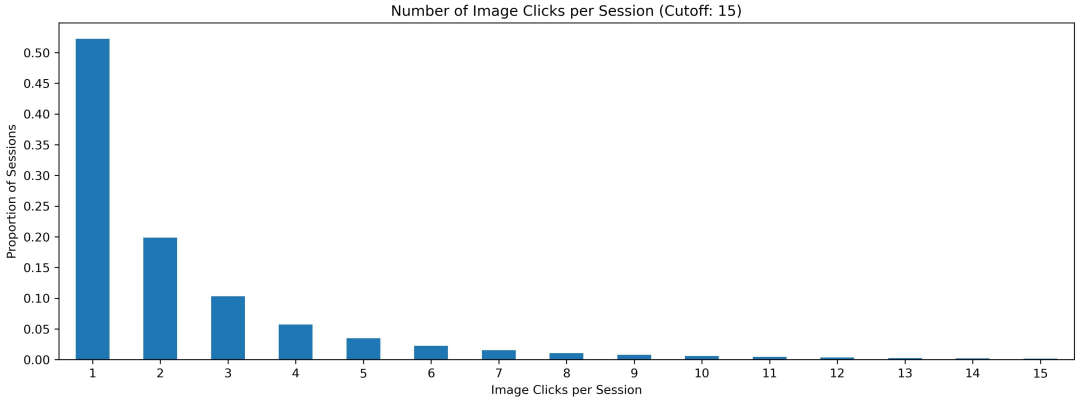


Figure 4.3: Visualization of number of IF image clicks in one session. The cutoff is set to 15 clicks.

The last column is the item ID which is a unique identifier of the respective IKEA product or IF image that refers to the corresponding action in that row.

The original raw dataframe also contains more detailed information about where exactly on the website a customer clicked on an IF image, since there are multiple ways or locations to start interacting with the feed. These information might provide further potential for improvements especially with regard to the reward design in the RL part. Nevertheless, they will be excluded from this work for the sake of simplicity and since a further investigation regarding the meaning and actual influence of this property would be necessary.

The raw data is almost entirely complete and only contains missing data in the `item_id` column. Here, 0.3% of values are missing, whose corresponding rows will simply be deleted since they represent such a small proportion of samples.

4.2 Data Preprocessing

The full raw dataset is large in size with over 73 million observations. Therefore, to create the preprocessed version of it, it is loaded in multiple JSON chunks, preprocessed and then concatenated to one single final dataset.

As a first step, a few column names are replaced and a unique session ID is assigned based on the three index columns for the session `start_time_ms`, `visitor_id`, `market`, to have a single key identifying a certain session.

In a second step, a numeric reward signal is added to each row according to the chosen reward structure. More details on this process and the reward structure used for the experiments can be found in Chapter 5.

The final step consists of first dropping all duplicated rows. Then, columns which are not relevant for creating the replay buffers, are dropped as well and only `sessionID`, `item_id`, `action_type`, `market`, `reward` are kept . It is important that this step is only done after the duplicates were already dropped, since otherwise, a lot of rows would be excluded, as the timestamp column gets removed. Customers tend to click on a product multiple times during a session, for example to compare it with other products, which would be considered as duplicates with the missing associated timestamp. After that, the item ID column is processed further, since in some samples instead of one ID, accidentally multiple IDs were added. These rows must be excluded because each of these collections of IDs in the item ID column would receive their own token in the tokenizing process, which is not desirable and would lead to an artificial bloating of the model with wrong and unnecessary parameters. Finally, the rows containing missing values for the item ID will be removed, returning a clean dataset that can now be split. For the full set, roughly 0.68% of rows are removed, yielding a body of 73,260,311 samples in total. In Table 4.2 the raw example from above is continued and shown after all preprocessing steps are applied.

Table 4.2: Example extract from the preprocessed data format

	sessionID	market	action_type	item_id	reward
0	0_6	ae	view_item	50298418	0.0
1	0_6	ae	view_item	50178411	0.0
2	0_6	ae	view_item	50178411	0.0
3	0_6	ae	view_item	50178411	0.0
4	0_6	ae	view_item	30488966	0.0
5	0_6	ae	click_insp	ee7f1af0-b8a...	1.0
6	0_6	ae	click_insp	879be313-4ab...	1.0
7	0_6	ae	click_insp	5afa5cdb-0f3...	1.0
8	0_6	ae	click_insp	b4dcec9d-d5f...	1.0
9	0_6	ae	click_insp	5963d9cc-99e...	1.0

4.3 Data Split

The clean dataset is then split into training, validation and test set according to the ratio 80%, 10%, and 10%. It is important to mention that these percentages refer to the number of sessions rather than the number of rows, since it is crucial that samples belonging to the same session will end up in the same dataset and in the exact same order they occured in the original set for creating the states in the replay buffer that represent the last interacted items or images in the right order. This is achieved by first extracting all the unique session IDs to a list and then shuffling them with a set random seed for reproducability to ensure a random ordering among the session IDs. Then, IDs are sampled according to the stated ratios to retrieve them from the dataset and to add them to the associated splitted set. This procedure does not lead to a row-exact split according to the specified percentages. Nevertheless, since the full dataframe is reasonably large and the distribution of number of clicks per session is fairly concentrated, the effects of longer and shorter sequences will balance out. Thus, when rounded, the split percentages are met.

The three datasets represent the basis for all the experiments. However, before being fed to any model, some further steps need to be taken to create the final replay buffers used for training and evaluation. This process will be discussed in more detail in Chapter 5.

5 Methodology

The following chapter will outline the methodology employed in this work. As a first part, the Supervised and RL definition will be applied to the given use case of training a RS for IKEA's IF, defining the problem in terms of the theoretical elements introduced in the chapters before. Then, building upon the previous data chapter, the replay buffer creation is illustrated. Finally, the application of the three different models - GRU4Rec, SQN and SMORL - including parameter settings and the corresponding evaluation procedure will be discussed.

5.1 Problem Definition

The following section describes the specific use case of IKEA's IF, first from a Supervised Learning perspective, followed by an exploration from a RL standpoint.

5.1.1 Supervised Perspective

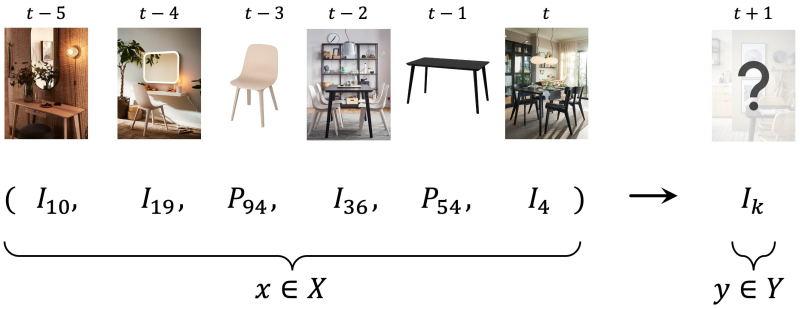


Figure 5.1: Supervised perspective problem overview. I is an IF image click and P a product interaction (for simplicity the kind of product interaction e.g. view/add-to-cart/etc. was omitted). Here, the window size for prior history is six.

As introduced in Chapter 5, Supervised Learning approaches try to learn the relationship between a body of input variables X and the corresponding known body of output variables Y, representing samples from the real underlying distribution $p(x, y)$, with the help of a sta-

tistical model. The goal is to be able to predict the labels Y with the help of the learned model, when only the input X is known. For the given use case, the input samples $x \in X$ represent the sequence of prior interactions with the IKEA website up to the current time step t while the outputs $y \in Y$ denote the IF images that were clicked in the immediate next time step $t+1$. This concept is depicted in Figure 5.1.

As explored in Chapter 4 describing the data, over 50% of all sessions only include one IF click. Longer click-sequences of only IF interactions are very rare. This means in most cases, customers will click on an inspirational image after viewing or interacting with products before. This characteristic must be incorporated into the problem definition, as exclusively modeling prior IF clicks would be ineffective and would likely result in unsatisfying performance of the RS in the real world. Therefore, it is essential that not only the IF but also the product interactions are included into the historic sequence represented by x. As shown in Figure 5.1 both, the product as well as IF interactions, are incorporated into x.

The corresponding labels $y \in Y$ contain the clicked image for $t+1$ and do not include products. This aligns with the RS's purpose to predict interesting images for the customer given their prior action sequence when they navigate to a page designed to display IF images. Following these characteristics, the output of the designed model needs to be a probability distribution over the corpus of all possible IF images. Thus, building on the theory in Chapter 2.1, this problem can be identified as a classification problem with the IF images representing the classes.

5.1.2 RL Perspective

As introduced in Chapter 2.4, the core of every RL problem definition is the MDP. For the application in the context of a RS for IKEA's IF, the tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$ according to the underlying finite MDP can be defined as follows:

- $\boldsymbol{\mathcal{S}}$: Discrete state space containing all possible states s, where s represents a sequence of previous user interactions with items or IF images of length n
- $\boldsymbol{\mathcal{A}}$: Discrete action space containing all possible next images to recommend
- $\boldsymbol{\mathcal{R}}$: Reward space as a subset of $\mathbb{R}$ containing all possible reward signals
- $\boldsymbol{p}$: Dynamics of the environment represented by $p(s', r|s, a)$ with s as current state, a the action taken in s, s' the next state and r the observed reward
- γ: Discount factor for the given use case

State Definitions, Actions and Environment

The state s contains information about the current status of the environment. It is the sequence of previous interactions of the user with the website, i.e., the same that was defined for x given the Supervised Learning perspective. The length of this previous interaction window is a design choice. For samples where there are insufficient prior interactions to completely occupy this window, any remaining positions will be supplemented with padding tokens. Given this definition, the state space $\mathcal{S}$ is both discrete and finite, since it contains all possible interaction windows of the defined size composed of sequences involving product or IF image interactions. It is obvious that, given that the training set alone contains over 120,000 different products and over 6,000 images, the state space is vast. Assuming a window size of u and ignoring the padding tokens, the number of possibilities for the stated amounts can be approximated by $120,000^n$. This term is exponentially increasing depending on the number of considered prior interactions. In Chapter 2.4, the Markov property was introduced, which requires the state to encode all necessary information from the past so that the decision for the next action given the current state is independent of the previous states and actions. This

property serves as a fundamental precondition for most RL algorithms. However, methods employing approximation techniques like DQN or DDQN do not heavily rely on it, as outlined in [30]. Despite this, the Markov property remains an important consideration when defining states for a given problem. In the context of this study, it is not immediately apparent how many preceding interactions should be included in the state to satisfy, or at least nearly satisfy, the Markov property. Consequently, all subsequent experiments involving RL will employ a history window of size ten for prior interactions, aligning with the approach utilized in the original SMORL and SQN paper [29], [39].

Another important consideration is the design of the next state s' that follows after the agent is in state s and takes action a. For this, we will first take a look at the environment itself, as the next state represents the changes to the agent's current state as a reaction to the taken action. As shown above, for IKEA's IF the action space $\mathcal{A}$ consists of the set of all IF images. Important to note in this context is that the actions must be interpreted from the agent's perspective rather than from a user perspective. They represent the image that the RS shows to the customer. The reaction of the customer to this image can be interpreted as the reaction of the environment to the action taken by the agent - they could decide to end the session or to engage further with products or images, yielding different reward signals for the RL system. Therefore, the underlying environment dynamics $p(s', r|s, a)$ are represented by the customer's behavior and are consequently highly stochastical. This is because different people might react entirely different to a recommendation and even the behavior of one and the same person can be deviant from one session to another based on their current interest, life situation or season of the year. Given this definition, the most theory consistent method to design the next state s' is to incorporate all interactions of the user after action a was taken by the agent in s until the next relevant state is reached. In this specific use case this state would be the next time the customer lands on a page designed to display IF images and clicks on one of them. This could be the immediate next step, assuming that they click on another IF image directly after viewing the image before. However, the customer could also interact with different products (e.g. view them, add them to the cart, etc.) before clicking another IF image. Based on the introduced theory and definition of the problem, consequently, all of these interactions as well as their connected rewards should be included in the next state and reward, until a new IF image is clicked or the session ends. This type of state definition will be referred to as *SD-all* in the rest of this work.

One problem with this definition could be that these interaction-chains happening between two image clicks can be very long and could therefore easily exceed the prior history window of 10. If this is the case, it will be very difficult for a RL system to pick up the relationship between the state and the following next state. Even though this definition is consistent with theory as the next state s' of one sample in the replay buffer consisting of s, a, r and s' will match the starting state s for the following sample in the buffer, these possible issues need to be considered as well. Moreover, this presumption is in line with experience of prior experiments conducted by IKEA in the context of RL applications for IF recommendations. These past experiments revealed that only incorporating the clicked image a recommended by the agent in s and the immediate next interaction, either product or image, showed more promising results. This state definition will be referred to as *SD-2* in the following parts.

One last alternative to the two presented possibilities, that will be covered in the experiments as well, is the *SD-1* definition. For this, only the clicked image represented by a is incorporated. From a practical perspective this means that the next state s' will contain the last nine interactions included in s plus the clicked image a. This definition is in line with the one used in the original papers [29], [39]. However, it needs to be pointed out that for their use case this definition is natural, as their dataset only contains products. Therefore, the set of sequence elements for the state and the action space share the same set of items, i.e., no intermediate interactions by the customer can occur in contrast to the use case at hand. Still, to keep close to their research, this state definition will also be part of the experimental set up for this work.

Reward

The reward r, as in every RL problem, is a signal that the agent receives after taking an action in a given state of the environment. As explained before, the choice of state definition will have a significant influence on the rewards that are assigned to a state and action pair s, a. In the SD-1 case, only the reward connected to the image click, represented by a, will be used. For SD-2, this reward plus the reward connected to the following interaction, either product or image, is applied. For the SD-all definition, the reward of all product interactions happened after clicking the image in a until the next image click will be summed up to retrieve the reward connected to the pair s, a.

The general reward structure is kept simple and is only dependent on the type of interaction. in addition, these definitions also represent values that have proven to be beneficial for IKEA in the context of training models in the involving RL. The specific reward signals assigned to each of the action types are presented in Table 5.1. As mentioned in Chapter 4, the first two entries in the table correspond to IF clicks and the other four to product interactions.

Table 5.1: Reward signals for different interaction types

Action Type	Reward
click_inspiration	2
select_content	2
add_to_wishlist	1
add_to_cart	2
purchase	2
view_item	0

Negative Reward Signals

An additional aspect worthy of mentioning within the RL perspective for the specified use case is the lack of negative rewards. As highlighted by Xin et al. [39], applying RL in the context of RS presents a unique challenge due to the exclusive presence of positive interactions in the data. In traditional RL environments, the agent receives both negative and positive learning signals, whereas in the case of the RS, only positive actions are recorded. This means that the data only contains samples in which the agent presented a recommendation to the customer and they actually clicked on it, as this is what is registered by the clickstream data collection procedure. However, there might exist many instances in which the customer lands on a subpage that displays IF images, i.e., recommendations by the agent, yet they choose not to engage with them, proceeding with searching or clicking on other items instead. These samples represent valuable learning opportunities for the agent, as it could learn which recommendations did not lead to a user click given their state s to reduce their output probability or the state-action value connected to it.

Summary of RL Characteristics for the given Use Case

The characteristics of the specific RL problem applied to the IF described above and the implications of the Q-Learning based models used for the experiments lead to the following characteristics of the method:

- **Model-free:** The dynamics of the environment are unknown and are represented by the underlying data distribution.
- **Episodic:** Each user-session represents one completed episode with a clear terminal state at the end.

- **Off-policy:** The behavioral policy π_b used to sample trajectories is encoded in the data samples and it differs from the learned policy π_o.
- **Offline:** All interactions with the environment are included in the dataset and the agent will not have the possibility to interact with it in an online fashion, i.e., to sample new trajectories.
- **TD-Learning with function approximation:** Q-Learning, which is based on TD-Learning, is used together with neural networks as function approximators. This also implies the usage of bootstrapping as defined in Chapter 2.4.

As discussed in Chapter 2.4, these characteristics include all three elements of the *deadly triad*. Consequently, there is an amplified risk for the emergence of diverging estimates and general instabilities in the training process, given the demonstrated properties of the problem at hand.

5.2 Creation of Replay Buffers

In the previous section, the use case of training a RS model for IKEA's IF was introduced from a supervised as well as a RL perspective. This section will present the steps necessary to create the replay buffers based on the preprocessed and split data demonstrated in Chapter 4.

The concept of the replay buffer is introduced in Chapter 2.4 in the context of DQN. It contains samples of state s, action a, corresponding reward r and the next state s' that followed after taking action a in s. That means that the preprocessed dataframe shown in Table 4.2 will be transformed to a dataframe containing the state, action, reward, next state and an indicator for whether this sample marks the end of an episode. One replay buffer is built for each of the three splitted datasets: train, validation, and test.

For each individual session, the following steps are performed:

1. **State:** For each of the IF image clicks in the session, build the corresponding state representation as a list containing the ten prior product interactions or image clicks. Save it as s. If there are less than ten previous interactions, pad the sequence by appending padding tokens to the *end* of it until state length is reached.
2. **Action:** Save the IF image click as a.
3. **Next state:** Build the next state representation depending on the chosen state definition and save it as s':
    a) **SD-1:** Delete the oldest interaction and replace it with the image click in a. If state contained padding tokens, replace one of the tokens with a.
    b) **SD-2:** Delete the two oldest interactions and replace them with the image click in a and the immediate next interaction in the session. Treat padding tokens as explained above.
    c) **SD-all:** Take the starting state of the following next image as next state. If there is no following image click, take the last ten interactions of that episode[1].
4. **Reward:** Calculate reward for s, a, depending on the chosen state definition and save it as r:
    a) **SD-1:** Save single reward of an image click.

[1] s' of last clicked image is irrelevant anyways as it represents a terminal state and as explained in Chapter 3.2.3 the corresponding Q-value will be set to 0.

b) **SD-2:** Add single reward of an image click with reward of next interaction depending on its type.

c) **SD-all:** Add all rewards of all interactions (including current click a) until the next image is clicked or a terminal state is reached.

5. **End of session marker:** Boolean that is true for the last IF click in a session.

In addition to the information presented in the enumeration, two more columns will be added to the replay buffer, which are not relevant in terms of theory but serve a more practical purpose. For each of the states s and s', the true state length is added, which refers to the length of the sequence excluding padding tokens. This is a crucial aspect for the implementation as most models will be trained using PyTorch's `PackedSequence`[2], whose creation requires the length of the unpadded sequence.

The information in the replay buffer can then be used during training not only for the RL part but similarly for the supervised classification part, where the inputs x are represented by s and the real labels y are the actions a.

The process described in Chapter 4, applied to the splitted datasets, results in a training set containing over 5.8 million samples, while the validation and test sets consist of approximately 730 thousand samples each. The combined size of these sets is significantly smaller than the full corpus of 73 million samples. This reduction in size is due to the buffer only retaining actions related to the IF.

5.2.1 Tokenization

During the steps of creating s, s' and a illustrated above, the product and image IDs, currently represented by strings, must be converted into a numerical format. This is because the models utilized in this study require numerical inputs. Maintaining a record of this procedure is crucial for accurately mapping model inputs and outputs to the corresponding products and images when analyzing results or deploying a model. To ensure reproducibility, a custom *Tokenizer* class is developed, which stores the exact mappings used in the associated replay buffer and can be reloaded from disk later on.

As the sequence of prior interactions that is used as an input for the model contains products as well as images, but the output space exclusively includes images, two different Tokenizers will be used for inputs s, s' and for the output connected to action a.

The input Tokenizer is created based on the following two sets - the set of all 121,106 product IDs included in the training set and the 6,313 IF images found in the full dataset. The decision to incorporate all unique IF images across all three splits, rather than solely those present in the training set, is primarily driven by practical implementation considerations. However, as the difference in the number of images only accounts to nine images which are not included in the training set, this represents a negligible detail. It will lead to nine further vector representations in the embedding layer of the networks that will remain untrained and maintain their randomized status. Subsequently, each of the IDs is assigned to an integer index starting from 0. Besides that, a padding token is added, as well as a token for unknown inputs, which is important when testing the model on validation and test set, as they likely contain products that are not covered in the training set. In total, the input Tokenizer contains 127,421 mappings.

The output Tokenizer incorporates not only the IF images present in the full dataset but also includes all available IF images online at the present moment. This decision was made in consultation with IKEA and holds implications for model deployment, as the operational pipeline requires the entire IF set as output dimension. So instead of the 6,313 images in the data, the full corpus of 10,107 images is used. Apart from a minor increase in the calculation

[2] `https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pack_padded_sequence.html`

requirements for the model due to the increased output dimension and the corresponding increase in model weights, this does not affect training. The additional images not included in the training set will never be used as an action a, i.e., real label y. Thus, due to the Cross-Entropy loss in the supervised part of each of the architectures, in every update, their weights will be adapted in a way that their output probability gets pushed down. This leads to them never being recommended, which, however, also makes sense when considering that they are not included in the training data. The output Tokenizer does not introduce any additional tokens, as padding is solely applied to the input sequences and not to the output domain. Moreover, since all the IF images present in the training set are included in the output set, there are no unknown image IDs. This implies that the output Tokenizer covers all possible IF images without the need for additional handling of unknown IDs.

Given the two Tokenizers, all the image and item ID strings will be converted to their numerical index representation when creating the states and actions in the replay buffer.

5.3 Implications for Model Architectures

In the following, details about the model architectures used in the experiments will be presented.

As a result of the tokenization procedure, each of the models has an input dimension of 127,421 and an output dimension of 10,107.

As explained in Chapter 3, each of the models used in the experiments shares the same GRU based backbone. To allow for a fair comparison, the architecture of this backbone will remain consistent across all models. The configuration is adopted from the original paper [29]. More specifically, the backbone will be a one-layer unidirectional GRU with a hidden dimension of 64.

For the embedding layer, in line with [29], a dimension of 64 is used as well.

5.4 Evaluation Framework

The procedure used for evaluating the different models trained in the experiments conducted in this work is closely related to the one used in the SMORL-paper [29]. All the metrics described in the following will be measured on both, the training as well as the validation set.

5.4.1 Accuracy

For the accuracy metrics, HR@k as well as NDCG@k will be calculated. As explained in Chapter 1, IKEA currently uses the top-12 predictions of the RS to show to the customer. Consequently, to align with IKEA's operational settings, both metrics will be computed not only for the top prediction but also for the top-6 and top-12 recommendations. So k is set to k=[1,6,12].

5.4.2 Diversity

Following the approach that the authors took in the original paper [29], CV_{div}@k is used to measure diversity. Similar to accuracy, for all experiments k=[1,6,12] is utilized.

5.4.3 Novelty

The metric for novelty is also adopted from the paper [29] and adjusted to the given data of the IF.

For the use case in the context of the IKEA IF, the possible item-space for recommendations is comprised of all existing inspirational images. Figure 5.2 shows the click-count

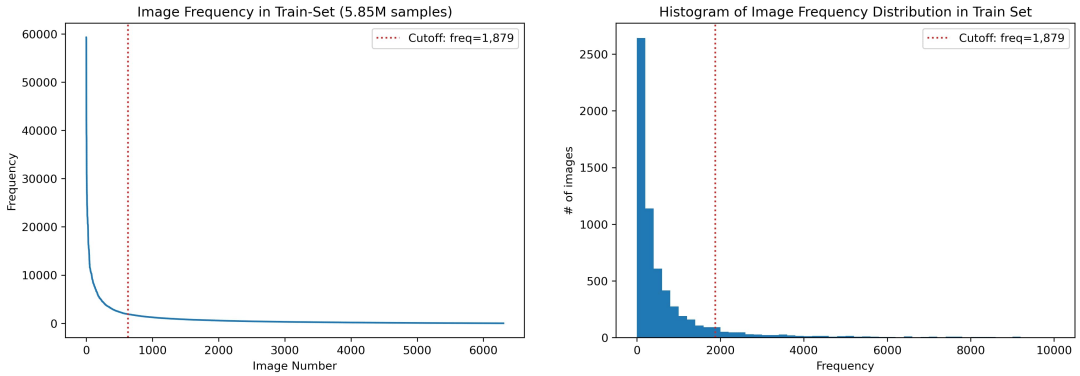


Figure 5.2: Left: Frequency plot of inspirational images included in training dataset. Right: Histogram showing the distribution of image frequencies. Cutoff lines refer to 90% quantile based on frequency distribution.

distribution of all 6,304 images included in the training set containing a total of 5.5 million interactions of customers with the IF (one interaction representing one click on an inspirational image). The plot on the left displays the frequency of image clicks, organized by the count-sorted image number. It clearly shows the long-tail phenomenon described earlier. A small number of IF images are very popular, with up to 60 thousand clicks. However, the biggest part of them is far less popular. As indicated by the 0.9-quantile cutoff-lines, which is calculated based on the frequency distribution, 90% of images were clicked less than 1,876 times. In the SMORL paper, the authors used this 0.9-quantile threshold to extract the unpopular items from the long tail and to use them for the novelty evaluation [29]. As we can see from the left plot in Figure 5.2, the cutoff line roughly marks the beginning of the long-tail, representing an appropriate threshold for extracting the unpopular image set. Therefore, similar to the paper, the 10% most popular IF pictures will be excluded and the remaining 5,672 are used for evaluating novelty coverage CV_{nov}@k. Similar to accuracy, for all experiments k=[1,6,12] is applied.

There is one more important point to note about both, the novelty and the diversity evaluation metric. As both of them are based on coverage and the set of unpopular images used for novelty is a subset of the full corpus of IF images used for diversity coverage, it is likely that both metrics will be positively correlated. This fact needs to be taken into account for the following analysis of experimental results.

5.4.4 Repetitiveness

As illustrated in the original paper [29], another important factor associated with a good RS is repetitiveness. There, repetitiveness is computed as the average number of repetitions in the top-k predictions per session included in the evaluation set. For this work, we will use a different way to approximate the repetitiveness-behavior of the model. There are several reasons for this:

Firstly, applied to the two kinds of repetitions that were identified in IKEA's IF in Chapter 1.2, the proposed metric approximates the repetitions associated to the red and blue images in Figure 1.3, i.e., duplicates inside recommendations from one timestep to another. However, it is not affected in any way by the repetitions highlighted in green, so when an image that was already clicked is shown as a recommendation once again. Given that this latter type of repetition was identified as having a greater potential to negatively impact the customer experience, particularly in the context of IKEA's IF, it would be advantageous for the metric to capture this behavior instead of the former.

Secondly, the proposed metric in the paper requires to keep track of all top-k predictions for each sample included in the evaluation set, which is highly memory inefficient, especially given the large size of the dataset underlying the following experiments.

Thirdly, it is debatable, if the measure appropriately captures the repetitiveness-characteristic of the trained model. As explained above, the authors compute the number of repetitions for each session in the evaluation set by substracting the amount of unique item IDs in the list of accumulated top-k predictions from the length of the entire list (equalling session length * k). The entire Python function, used to calculate repetitions, from the paper's original code base can be found in Figure A.10 in the Appendix.

Table 5.2: Small example replay buffer with state length three. Numbers refer to item IDs.

State	Action	Next State
[3,9,4]	11	[9,4,11]
[9,4,11]	2	[4,11,2]
[4,11,2]	19	[11,2,19]

Let us look at a small sample session to understand why this evaluation strategy could lead to a problem. Table 5.2 shows a simple example excerpt from a replay buffer for an example session. The state column contains the item IDs of the products the customer was in contact with before and the action refers to the next interacted item given the state. Let us further assume that when the data was collected, the customer had eight different next items as recommendations. To compute the repetition score for this session, for example for k=5, each of the three states would be plugged into the trained model and each of the five highest ranked next item predictions yielded as outputs by the model would be saved. This procedure works fine if the recommendations of the newly trained model are very similar to the ones yielded by the model that the recorded customer data is based on. So if the real next item is included in the top-8 predictions in this case, i.e. if this trajectory of states and actions is in the set of possible trajectories given the model. But what happens if this is not the case? Let us assume, when inputting the first state [3,9,4] into the model, the top-8 predictions were [10,17,1,6,12,21,7,19]. In this case, the real item interacted with by the customer, item 11, was under the top-8 recommendations from the old model, but is not included in the top-8 predictions from the newly trained model. This means that the trajectory recorded and saved in the replay buffer does not represent a possible trajectory given the new model, since the customer would have only had the eight options stated before. This could have led to an entirely different trajectory with different states and possibly deviating repetition behavior. Nevertheless, for calculating the repetition score, the next state included in the replay buffer, so [9,4,11], would be plugged in as the new customer state to get the next top-5 predictions. This could possibly lead to a wrong or misleading assessment of repetitiveness, especially when the two models differ significantly in their prediction behavior.

In addition, it is important to mention that this does not pose a major problem for the other metrics since they are based on measures that ignore the concept of sessions and are independent of the order in which samples from the replay buffer are processed. However, for repetition, the concept of session is kept accumulating all top-k predictions based on the temporal sequence of customer interactions in this session. Still, it needs to be kept in mind, that the accuracy metrics suffer of a bias introduced by the behavioral model used to record customer interactions. This is because, firstly, it is unknown, whether the state s in the replay buffer is a possible or reachable state given the new model. Secondly, the customer's decision only reflects the optimal choice based on the number of recommendations presented to them on the website. Thus is does not represent their true preference in the current state if they had the choice among all possible items.

In general, the metric included in the paper will still yield a good approximation for the repetitiveness of the model. However, due to the presented reasons, an adapted metric will be used in the experiments of this work, which drops the concept of sessions and is entirely

based on the interacted items included in the customer state. It measures how many repetitions on average the model yields in its top-k predictions based on the fixed length prior interaction history in the state s. This way, the temporal order of item interactions inside a session gets less important and it is only measured, whether an item that the customer already interacted with in the last, most current timesteps included in the state, is repeated in the top-k recommendations. The adapted metric is calculated as

$$R@k = \frac{1}{N} \sum_{i=1}^{N} |\{g : g \in \text{topk}_i \text{ and } g \in s_i\}|.$$

Here, N is the number of total item interactions (i.e., state-action pairs) in the dataset, topk_i represents the top-k predictions for the sample with the according state s_i. The state s_i is the vector holding all prior u interactions for sample i. Ignoring the RL notation, it can also be referred to as the general input x_i to the models, in line with the notation used in Chapter 3.2.

As a matter of fact, this procedure ignores possible repetitions that could occur from one recommendation-batch in a certain state to the next recommendation-batch in the next state and only captures repetitions based on the current state and recommendation-batch. However, as discussed above, these kinds of repetitions were identified to be more relevant for IKEA's IF, which is why this metric is more suitable for the given use case.

Furthermore, it is important to note that this new metric will be dependent on the state length of the replay buffer. For longer states, i.e., a longer window of item interactions, it is more likely for the model to repeat one of the items included in it compared to very short sequences with only a couple of interactions. Thus, this metric may only be compared among models using input-states of the same size.

Similar to accuracy, for all experiments the introduced repetitiveness metric R@k will be measured for k=[1,6,12]. However, this metric should be viewed as yet another way of getting an insight into the potential behavior and recommendation characteristics when interacting with customers. It should only be used for an additional information gain rather than for selecting a model. For these purposes the provided accuracy metrics offer more valuable insights. It is easy to show that the metric might give misleading hints: Let us think of a general deterministic RS model for next item prediction, that always chooses the exact same item as its top recommendation for the next step. In this case, as this item is unlikely to appear in the user's recent interactions, the R@1 metric would nearly always be zero, implying a non-repetitive model. At the same time it would also have an HR@1 value of almost zero, as recommendations are deterministic and (in most cases) incorrect. In contrast, a well-trained model that adjusts its recommendations based on the user's click history might recommend items similar to those the user has already interacted with. Consequently, this model is more likely to duplicate items, leading to a higher R@k score. Therefore, while analyzing the results, it is crucial to consider this constraint.

5.5 Novelty and Diversity Reward

The novelty reward for the SMORL model as discussed in Chapter 3.2.4 is based on the same set of unpopular items as the novelty metric itself, which is why its application is straight forward and does not need to be investigated further.

For the diversity reward however, this is different. It translates the cosine similarity of the top predicted image and the last interacted item or image in the historic sequence in s to a reward. This process relies on the pre-training of an embedding layer. The corresponding embedding matrix is then used to retrieve the embedding vector of the product or image, which is the basis for the calculation of the cosine similarity. For this purpose, following the recommendations in the original paper [29], a GRU4Rec model is trained and its embedding layer is extracted and used throughout all the experiments to calculate diversity rewards on

the training as well as on the evaluation datasets. Further details on the procedure as well as a thorough analysis of the yielded embedding results will be presented in the beginning of the following chapter.

6 Results

This chapter first analyzes the pre-trained embeddings used for the diversity reward, then presents the results of the comparative analysis of GRU4Rec, SQN and SMORL and lastly explores the effect of different window sizes for the input of the models, representing the sequence of prior customer interactions.

All of the models included in the experiments as well as the evaluation framework are implemented from scratch in Python using the deep learning library PyTorch. Each of the following experiments is run on the Google Cloud Platform using a virtual machine with an N1 high-memory CPU with 104 GB RAM and an NVIDIA P100 GPU with 16 GB VRAM as accelerator.

6.1 Embedding Pre-Training and Analysis

As explained in Chapter 5, to compute the diversity rewards during training of SMORL and during evaluation in general, a pre-trained embedding matrix is required [29], that originates from a GRU4Rec model run. Thus, before the main experiments can start, a GRU4Rec model is trained on the training set of state size ten, with the previously described GRU-backbone from Chapter 5.3. The embedding dimension is set to 64, in line with the original paper [29]. The learning rate is set to 0.0005 and the model is trained for eight epochs. The final extracted embedding parameters originate from epoch seven which showed the best NDCG@12 on the validation set. As explained, these frozen embeddings will be used throughout all of the following experiments for diversity reward calculations.

In order to ensure that the embeddings have successfully captured meaningful vector representations of both images and products, a thorough analysis is conducted in the following. This analysis not only verifies that the rewards calculated for the SMORL diversity head are reasonable but also offers further important insights. As the GRU4Rec model itself is a model that will be compared and as it shares the same backbone structure with SQN and SMORL, this analysis is also relevant to the general question of whether the embedding layer is learning effectively. This aspect is of particular importance as the vector representations form the fundamental groundwork for all subsequent computations within the GRU, as well as further operations within the respective heads of the different models. Therefore, it is a key element in the architecture whose proper functionality will be studied in the following.

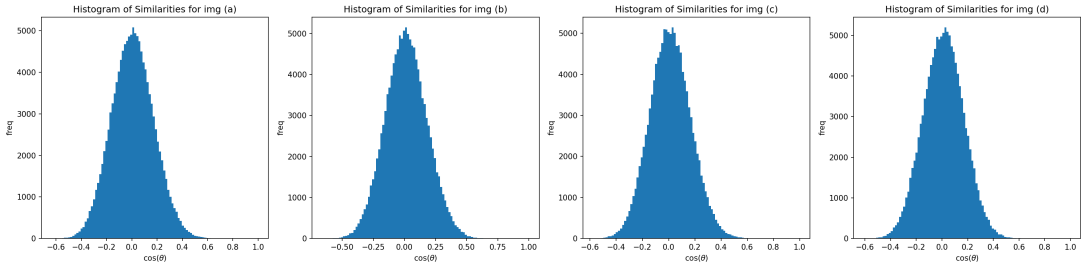


Figure 6.1: Distribution of cosine similarity for the four IF images investigated in Figure 6.2 and all other items and images in the input space.

For the analysis, different example IF images are chosen and transformed into the saved 64-dimensional vector representations included in the trained and extracted embedding layer. The goal is to identify the most similar and dissimilar images or products by computing the cosine similarity between the IF embedding vector and every other vector present in the embedding matrix. Four IF images are chosen for the analysis. Figure 6.1 shows the distribution of the cosine similarity between each of the images and all 127,421 products and images in the input space, including the images themselves. We can see that all distributions are very similar and symmetrical, resembling a normal distribution with mean around zero. This means that many products and images are not very similar nor very dissimilar in terms of cosine similarity. As the values become more extreme in either direction, they become increasingly rare. While it might be difficult to analyze exactly why the process results in the presented distribution - given that it essentially measures the cosine of the angle between two vectors in $\mathbb{R}^{64}$ - it does facilitate the identification of the most similar and dissimilar embeddings. Each image, therefore, has some distinct similar and dissimilar counterparts within the product-image space, according to their learned vector representations. These counterparts are the images and products that are located at the tail of the distribution, showing the extremes on each side. We can see that the range of values for dissimilarity extends to approximately -0.6 and the one for similarity ranges all the way up to one. The reason for this is that the corresponding image itself is included in the embedding matrix. This always results in a cosine similarity value of 1, given the comparison between identical vectors.

Based on the similarity scores computed in the step before, the most extreme ones for each of the example IF images can be extracted and qualitatively analyzed. Figure 6.2 shows the ten most similar and dissimilar products or images, based on their cosine similarity, for the four chosen example IF images. These images are highlighted in green and serve as reference points with a cosine similarity of 1. Products are excluded for the first three figures for a better overview. However, the final example considers both IF images and products. Inside each row, results are sorted by similarity score in ascending order.

The four chosen pictures are random selections among some of the most clicked pictures in the IF in the respective period included in the datasets. The first one in 6.2a shows a desk setup with a drawer cabinet. The second one in 6.2b just depicts a long white shelf that can be used in different locations in a house. The image in 6.2c shows a section of a youth room with gaming equipment and a desk. The last one in 6.2d illustrates a fridge filled with transparent food boxes, some vegetables and bottles.

As the first step of the qualitative analysis, we can see that the concept of similarity and dissimilarity is generally well captured. For instance, the images most closely associated with the desk setup in 6.2a share a stylistic resemblance, with all of them depicting different rooms featuring white desks. Conversely, the furthest embeddings predominantly belong to the kitchen or cooking context. The pattern holds true for the other three images as well, with similar images in terms of style and category being the closest, and those from contrasting categories being the furthest. For the shelf in 6.2b, the furthest images are largely outdoor

related, while for the gaming desk in 6.2c, they are associated with food or living rooms. For the fridge in 6.2d, the most dissimilar images predominantly feature bedroom furniture.

Another important observation is that the IF images being closest to the ones examined not only share the same category and style but frequently include products that are also present in the reference image. For image 6.2a the exact drawer cabinet can be found in multiple other examples and the image being closest to 6.2b shows the same shelf, once in black and in white. This gets even more apparent, when looking at the last example in 6.2d, which not only shows IF images but also includes product embeddings. The reference image of the fridge contains multiple of the transparent containers and the storage turntable "SNURRAD". These two products are shown among the top-10 most similar embeddings, which again confirms the effectiveness of the embedding process. More of such examples with products included can be found in the Appendix in Figure A.2.

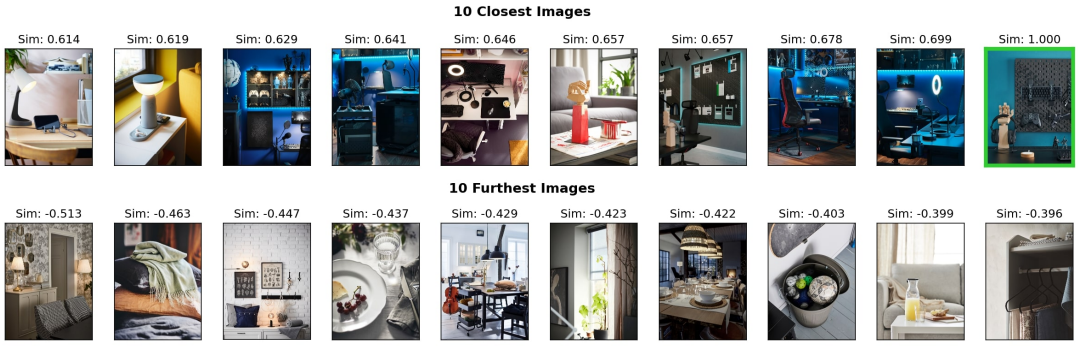
An interesting insight indicating that the embeddings capture more than just the image's category for determining vector proximity or distance can be can be seen when examining at 6.2c. There are two IF images of couches shown, one identified as similar and the other as dissimilar to the gaming desk picture. While the one being far away from the examined gaming desk picture features juice and breakfast on the table next to the couch, the other one shows a similar setting but with the same wooden decoration hand on the table as can be found on the desk in the original one. So even though the "living room" category per se does not seem to be closely related to the "youth-room" category of the original image, the embeddings still picked up the similarity of these two images during training as they both feature the same product.

The results of the examples shown are not just "cherry picked" but are representative as some further visualizations shown in Figures A.1 and A.2 in the Appendix reveal. One more specific example of these visualizations is picked to explain another important fact. The illustration shown in Figure A.1b presents the results for a dark blue commode. At first glance, the two closest images associated with the commode appear to be duplicates. However, upon closer inspection, they portray the same room setup but from slightly different angles. This is typical for the database of IKEA's IF images, as there might exist multiple images of the same room installation with minor changes to it. When qualitatively evaluating a trained RS for repetitions among predictions, showing both images would most likely be an undesirable behavior. However, for the model, these images, despite their similarities, have different indices and are therefore treated as entirely separate inputs or outputs. This also implies that the repetitiveness metric introduced in the methods chapter will miss this behavior, which is important to keep in mind for later analysis of the prediction results yielded by trained models. This behavior can be observed in the same example when looking at the two closest products instead. They both show the same dark blue commode, also featured in the reference image. This might be a result of the same item having two different product identification numbers connected to the same product due to IKEA's internal database design (e.g. for different markets). The same happens for different colors for the same product. However, there is still an interesting insight connected to both of the alleged duplicates: While for the trained model and thus also the trained embedding layer these duplicates are entirely different items and images, it still once again showcases that the model learned stable and consistent vector representations. If we compare the similarity of the two duplicate images (0.741 vs. 0.807) and the two commodes (0.622 vs. 0.642), the system learned embeddings that place each pair of duplicates very close together in the 64-dimensional vector space, as reflected by their near-identical cosine similarity scores.

However, it is important to note that there is still a potential risk when the same product, e.g., with the only difference being color, is represented by separate IDs in the training data. This risk is particularly significant for low-frequency items or variants of this item with low occurrence. As these items are rare in the dataset, it could lead to suboptimal or less robust learned vector representations for those specific items. As a consequence, this may lead to undesired recommendation behavior.

(a) Desk setup

(b) Long shelf



(c) Youth desk for gaming

(d) Fridge with storage boxes

Figure 6.2: Embedding visualization of IF images based on cosine similarity. Top row shows ten closest and bottom row ten furthest images. Example 6.2d also includes product embeddings. Sorted in ascending order with regard to similarity score. The image highlighted in green is the one used for comparison.

6.2 Comparison of GRU4Rec, SQN and SMORL

This section presents the experiments conducted for the comparative analysis of GRU4Rec, SQN, and SMORL. It includes a detailed discussion of the parameter settings, hyperparameter tuning performed on the validation data, and the final model evaluations for each of the three models. The obtained results from these experiments are subsequently utilized to compare the three models with regard to the different metrics in the evaluation framework.

6.2.1 General Model Settings and Evaluation

In line with the approach employed in [29], the history window size of the sequences in the states found in the replay buffer will remain constant at ten for all experiments. The common GRU backbone across the models is a one-layer unidirectional GRU with a hidden dimension of 64. Consistent with the aforementioned paper, the same dimension of 64 is used for the embedding dimension. The batch size, which refers to the number of samples from the replay buffer used to compute the gradient approximation for the parameter updates, is maintained at 256 for all the experiments, which is also adopted from [29]. For evaluation purposes, the batch size is set to 5,000. However, it is important to note that this value influences only model runtimes and does not affect any of the metrics calculated during the evaluation step.

Each model will be trained for a total of eight epochs, while every 25% of an epoch, the current model will be evaluated on the provided validation dataset, resulting in four evaluation steps per epoch. The metrics tracked for evaluation introduced in Chapter 5 are the following, which are computed both for the training and the provided validation set:

- Supervised- and Q-Loss
- Accuracy - HR@k and NDCG@k for all $k \in [1, 6, 12]$
- Diversity - CV_{div}@k for all $k \in [1, 6, 12]$
- Novelty - CV_{nov}@k for all $k \in [1, 6, 12]$
- Repetitiveness: R@k for all $k \in [1, 6, 12]$

In addition to that, for SQN and SMORL, which are based on double Q-learning as presented in Chapter 2.4, the performance of the second model is monitored as well. After each evaluation, all metrics are logged. During each run, the best-performing model, according to a predetermined validation metric, will be saved. The chosen metric for model performance, as agreed upon in consultation with IKEA, is the NDCG@12, as accuracy among diversity and novelty is still the most important objective for IKEA. Moreover, as discussed in Chapter 5, while the hit ratio provides easier interpretability, the NDCG@k is a better choice for model comparison as it also reflects the ranking in its calculations. The choice for k=12 results again from the specific characteristic of the IF, to show the twelve top-predictions of the model to the customer. Consequently, it is logical to base the determination of the best model with respect to the chosen accuracy validation metric on the top-12 predictions. After each evaluation step, the NDCG@12 on the validation set is compared to the highest one registered in the epochs before and if it shows an improvement, the parameters of the saved model are overwritten with these new ones. After all epochs are done, this process yields the model copy that contains the parameters that performed best during evaluation based on the validation NDCG@12. This helps to avoid overfitting of the model and is known as *Early Stopping* [9]. In the case of SQN and SMORL, where two distinct neural network copies are involved, the comparison to determine the best model with regard to th validation NDCG@12 metric considers both copies collectively.

Following the original paper implementation [29], the embedding matrix is initialized according to a normal distribution with $\mu = 0$ and $\sigma = 0.01$.

As optimization algorithm, the *Adam* optimizer, which stands for Adaptive Moment Estimation, is used. It is an advanced version of the stochastic gradient descent algorithm that additionally takes the first and second moment of the gradient into account and it is commonly used in various deep learning tasks. The exact details of the algorithm are out of scope for this work and the reader is once again referred to the book by Goodfellow et al. [9] or the original paper that introduced it [17]. When training SQN and SMORL, two separate Adam optimizers are used, one for each set of model parameters.

6.2.2 Model Comparison Procedure

To ensure a fair comparison across the three model classes, an initial step involves conducting a hyperparameter search. This process includes training models of each class on the training dataset using various hyperparameter settings. After that, based on the saved models for each run following Early Stopping, one representative for each class, GRU4Rec, SQN and SMORL, is chosen for comparison. The representative model is the one that displayed the highest NDCG@12 among its peers, thus marking it as a promising candidate for further examination. The parameter setting of the best model in each class is then used to train four models for each class with these exact parameters, which are then evaluated in a final step on the so far unseen test dataset, to allow for a conlusion regarding the comparison. The optimal parameter settings for each model class, derived from the best-performing model, are subsequently applied to train four separate models within each class. These models, trained with the same optimal parameters, are subsequently evaluated on the previously unseen test dataset. This final step enables a thorough comparison of the different model classes and an approximation of the influence of randomness on the training results.

It must be noted that the conducted hyperparameter search is by no means exhaustive as computational resources are limited. It is focused on the most important parameters like the learning rate and for SQN and SMORL, which include RL, the state definition and reward structure. Also, the architecture and settings of the backbone network as mentioned before will not be varied to keep experiments for different model classes comparable. The exact parameters of the conducted hyperparameter searches will be presented in the respective sections.

6.2.3 GRU4Rec - Hyperparameter Search

As the GRU4Rec model is independent of the state definitions since it does not include RL, the learning rate will be used as the main hyperparameter to vary. The model is trained under four different learning rate settings. Using the best learning rate of these runs, one more experiment is conducted targeting the way that the embedded input sequences are passed to and processed by the GRU layer. As discussed in Chapter 5, as a default, the implementation of the backbone uses PyTorch's `PackedSequence` class to pass the embedded sequences in the input to the GRU. This packed sequence can be computationally advantageous and is yielded by passing the embedded sequences together with their true lengths to the `pack_padded_sequence()` function[1]. The true length refers to the number of non-padding tokens in the right-padded sequence. The `PackedSequence` retains only the meaningful information from the non-padding sections of the sequence, which can result in more efficient memory use. However, another important feature of using the `PackedSequence` is that for the computations of the final hidden states yielded by the GRU after unrolling it over the sequence, only the non-padding vectors are considered. This means that the vector representation of the padding tokens included in the embedding layer - and therefore the padding tokens themselves - have no influence on any computations within the architecture. An alternative for feeding the sequences to the GRU is not to pack them and to feed the raw

[1] `https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pack_padded_sequence.html`

Table 6.1: GRU4Rec hyperparameter setup for experiments

Nr.	Name	Learning Rate	Packed Sequences
1	GRU4Rec-lr-01	0.01	True
2	GRU4Rec-lr-001	0.001	True
3	GRU4Rec-lr-0005	0.0005	True
4	GRU4Rec-lr-0001	0.0001	True
5	GRU4Rec-lr-001-trainpad	0.001	False

embeddings of all sequence elements to the model. This way, the padding tokens are considered in the calculations of the hidden states and can thus influence them. This leads to the fact that the gradient from the loss computations will also flow into the vector representation of the padding token inside the embedding layer, which allows it to be trained as well. This capability could be beneficial if, for example, there is a common pattern where customers are likely to click certain images at the start of a session, which would be represented in sequences with a higher count of padding tokens.

All conducted hyperparameter experiments for the GRU4Rec class are shown in Table 6.1. The other parameters that are not included in the settings are set to their default value discussed in the introduction of this chapter.

The validation metrics' progression across all eight epochs, equivalent to 32 evaluation steps, is illustrated in Figure A.3 in the Appendix. Each metric except for the loss is calculated three times based on the setting of k. However, for simplicity, only the ones computed on the top-12 predictions are showcased, as these were previously identified as the most relevant ones for the given use case.

The evaluation results on the validation set for using the model parameters from the best performing epoch with regard to validation NDCG@12 of each of the five runs is depicted in Table 6.2. For the loss, which corresponds to the Cross-Entropy loss, and for repetition R@k, a lower value is preferred. Conversely, higher values are favorable for all other metrics related to accuracy, diversity, and novelty.

Table 6.2: Validation results for GRU4Rec hyperparameter search

Nr.	Name	Loss	HR@12	NDCG@12	CV_{div}@12	CV_{nov}@12	R@12
1	GRU4Rec-lr-01	5.0533	0.4410	0.2511	0.5484	0.5185	0.9053
2	GRU4Rec-lr-001*	4.7082	0.4974	0.2907	0.5858	0.5584	0.9764
3	GRU4Rec-lr-0005	4.7415	0.4936	0.2895	0.5770	0.5490	0.9651
4	GRU4Rec-lr-0001	4.9402	0.4684	0.2757	0.4807	0.4462	0.9002
5	GRU4Rec-lr-001-trainpad	4.7040	0.4978	0.2912	0.5867	0.5593	0.9732

Note: * indicates hyperparameter settings chosen for testing

Experiment 1 that is using the highest learning rate shows an unstable learning behavior as observable in Figure A.3, indicating a too high learning rate. Experiment 4 on the other hand is characterized by a slow convergence, which suggests that the learning rate is too low. In Table 6.4 we can see, considering runs one to four, that setting the learning rate to 0.001 produces the most favorable results across all metrics with a HR@12 of 0.4974, indicating that the real image that the user clicked on is included in the top-12 predictions of the model in over 49% of the samples in the validation set. A slight improvement in all dimensions is observed in experiment 5, which maintains the same settings as experiment 2 but also incorporates the learned padding token representations into its computations. However, as

these improvements are minor and could also be due to randomness, the setup of experiment 2 is chosen as the final model for GRU4Rec to be used in the model comparison. This is due to the fact that as explained before the computational complexity of the model increases when the `PackedSequence` is not used. The marginal gains from using the learned padding token representations do not outweigh the increased computational demands.

The full table including all computed metrics for the five hyperparameter search runs based on the training and the validation data are shown in Table A.1 and A.2. A comparison of results from the training and validation datasets reveals a slight tendency towards overfitting. Specifically, the NDCG@12 value for the validation samples stands at 0.2907, while it climbs to 0.3130 for the training set. This highlights a potential area for further model improvement, for example through regularization techniques like dropout [28].

6.2.4 SQN - Hyperparameter Search

For SQN, the first model using RL, the hyperparameter search will similarly to the one for GRU4Rec be focused on the learning rate and additionally on the different state definitions introduced in Chapter 5.1.2. In this context, a total of five experiments is conducted. The first three use the traditional state definition SD-all for RL while the learning rate is varied. In the last two runs, the best performing learning rate from the first three runs will be used together with the other two state definitions, SD-2 and SD-1. An overview of the setups can be found in Table 6.3. The discount factor γ for the Q-learning part of the model is adopted from the original paper [29] and is set to 0.5. The other parameters are again set to the default values defined before.

Table 6.3: SQN hyperparameter setup for experiments

Nr.	Name	Learning Rate	State Definition
1	SQN-lr01-sdall	0.01	SD-all
2	SQN-lr001-sdall	0.001	SD-all
3	SQN-lr0005-sdall	0.0005	SD-all
4	SQN-lr001-sd2	0.001	SD-2
5	SQN-lr001-sd1	0.001	SD-1

The corresponding evaluation steps during training for all experiments are shown in Figure A.4 in the Appendix. An interesting observation from the depicted curves is that the models using the SD-all state definition show very different and overall undesirable training behavior. The two other models using SD-2 and especially SD-1 resemble the training process of the GRU4Rec models.

The most relevant metrics based on the validation data are presented in Table 6.4. These metrics confirm prior observations: models trained with the SD-all state definition generally perform noticeably worse across all metrics, except for repetitiveness. For instance, the second experiment, which is the best performing SD-all run, has a HR@12 score of 0.4024 compared to 0.4960 belonging to the last and best model given the validation data evaluated on.

In this context, the problem regarding the repetitiveness metric discussed earlier in Chapter 5.4.4 is highlighted. While the R@12 score of around 0.54 of experiment 1 indicates less repetitions compared to 0.95 in experiment 5, there is a big gap in all other dimensions. For example, experiment 1's novelty score is less than half of that of experiment 5. While the model in the fifth run manages to cover over 55% of the previously defined set of unpopular IF images, the model in the first run only covers around 20%. Given these outcomes, the model configuration for SQN selected for further comparison is derived from experiment 5.

Table 6.4: Validation results for SQN hyperparameter search

Nr.	Name	Loss	HR@12	NDCG@12	CV_{div}@12	CV_{nov}@12	R@12
1	SQN-lr01-sdall	6.5480	0.2021	0.1033	0.2530	0.2045	0.5357
2	SQN-lr001-sdall	5.2932	0.4024	0.2309	0.4774	0.4427	0.8331
3	SQN-lr0005-sdall	5.6405	0.3429	0.1908	0.2779	0.2300	0.7459
4	SQN-lr001-sd2	4.8227	0.4807	0.2814	0.5651	0.5362	0.9509
5	SQN-lr001-sd1*	4.7189	0.4960	0.2899	0.5836	0.5560	0.9749

Note: * indicates hyperparameter settings chosen for testing

6.2.5 SMORL - Hyperparameter Search

As in both of the previous hyperparameter searches for GRU4Rec as well as for SQN a learning rate of 0.001 yielded promising results, this learning rate is maintained for subsequent SMORL experiments. This decision is driven by the increased hyperparameter space that the model introduces. The SQN model compared to GRU4Rec features a RL part. Here, the underlying state and reward designs can be varied as well as the discount factor γ. For the SMORL model, however, as explained in Chapter 3.2.4, the parameters α and w are added. As a reminder, $w \in \mathbb{R}^3$ represents the weighting between the three loss signals from the different Q-heads in the calculation for the loss L_{SDQL} of the RL part, while α regulates the balance of L_{SDQL} and the supervised loss L_s for the computation of the unified loss signal L_{SMORL}. This way, by elevating α, the influence of the Q-learning heads on the model parameter updates can be increased.

Table 6.5: SMORL hyperparameter setup for experiments

Nr.	Name	State Definition	w	α	TCR
1	SMORL-sdall	SD-all	[1,1,1]	1	12
2	SMORL-sd2	SD-2	[1,1,1]	1	12
3	SMORL-sd1	SD-1	[1,1,1]	1	12
4	SMORL-sd1-NovDiv-alpha1	SD-1	[0,1,1]	1	12
5	SMORL-sd1-NovDiv-alpha5	SD-1	[0,1,1]	5	12
6	SMORL-sd1-NovDiv-alpha5-TCR1	SD-1	[0,1,1]	5	1
7	SMORL-sd1-NovDiv-alpha100-TCR1	SD-1	[0,1,1]	100	1
8	SMORL-sd1-NovDiv-alpha150-TCR1	SD-1	[0,1,1]	150	1
9	SMORL-sd2-alpha5-133-TCR1	SD-2	[1,3,3]	5	1

TCR = # top recommendations considered for Nov./Div. reward calculation

In their original paper, the authors argue that *"the SDQL loss is dominated by self-supervised loss, which suggests that the optimization of parameter α [...] might improve the effect of SMORL part on the base model"* [29]. They compare values for α in the interval of $[1, 10]$. Given the dataset used for the experiments, specifically for the SMORL model using a GRU backbone, they found that $\alpha = 1$ yielded the most promising results. This is why this value will be adopted for the first three hyperparameter experiments, that will explore the effect of training SMORL based on the three distinct RL state definitions. All hyperparameter experiments are presented in Table 6.5.

The parameter TCR in the setup table is specifically introduced for this work and it refers to the number of top recommendations that are used for the calculation of the diversity and novelty reward. As a reminder, the diversity reward for the SMORL model, as presented in Chapter 3.2.4, is based on the cosine similarity of the most recent interacted item in the click-history of the customer and the top prediction of the model for the given user state. The same is true for the novelty reward calculation, which is similarly based on the top prediction yielded by the model. In this context, the authors stated the following: *"Basing the diversity reward system on top prediction p_t and top-k recommendations instead of only the last clicked item l_t was considered, but we observed no improvement in performance"* [29]. Considering the unique characteristics of the given use case from a theoretical perspective, it appears reasonable to base the reward calculations not solely on the most likely next image recommended by the model, but rather on the top-12 predictions. This way, the model would receive a higher reward if the average cosine similarity between the last interacted image or item in the prior history and the top-12 recommendations is smaller, indicating a more diverse batch of predictions. Similarly, for novelty, the reward will be higher, if on average more of the top-12 predictions belong to the unpopular set of long-tail images that was previously defined. This use-case specific adaption of the reward design is used throughout the first five experiments.

Table 6.6: Validation results for SMORL hyperparameter search

Nr.	Name	Loss	HR@12	NDCG@12	CV_{div}@12	CV_{nov}@12	R@12
1	SMORL-sdall	5.2896	0.4034	0.2313	0.4619	0.4261	0.8376
2	SMORL-sd2	4.8088	0.4820	0.2818	0.5605	0.5314	0.9554
3	SMORL-sd1*	4.7053	0.4984	0.2914	0.5820	0.5543	0.9733
4	SMORL-sd1-NovDiv-alpha1	4.7074	0.4980	0.2909	0.5810	0.5532	0.9764
5	SMORL-sd1-NovDiv-alpha5	4.7148	0.4974	0.2909	0.5683	0.5397	0.9709
6	SMORL-sd1-NovDiv-alpha5-TCR1	4.7268	0.4954	0.2900	0.5631	0.5341	0.9815
7	SMORL-sd1-NovDiv-alpha100-TCR1	5.4513	0.3772	0.2151	0.3161	0.2708	0.7837
8	SMORL-sd1-NovDiv-alpha150-TCR1	5.5737	0.3567	0.2028	0.3294	0.2848	0.8089
9	SMORL-sd2-alpha5-133-TCR1	5.1637	0.4240	0.2470	0.4940	0.4604	0.8850

Note: * indicates hyperparameter settings chosen for testing

Let us begin by taking a look at the first three experiments targeting the different state definitions. The plots for the validation curves in Figure A.5 in the Appendix, as well as the most important validation metrics measured on the best performing model for each run shown in Table 6.6, demonstrate that, just like with SQN, the SD-all state definition underperforms compared to the other two across all metrics, except repetition. The best one of the three was again the SD-1 model with an NDCG@12 of 0.2914. This is why for most of the following experiments, this state definition is kept.

Based on these findings and following the approach outlined in the paper [29], the next set of experiments will focus on investigating the impact of the newly introduced parameters w and α. Specifically, the aim is to analyze the influence of the two RL heads for diversity and novelty, which were added in SMORL compared to SQN. In experiments 4 to 8, the accuracy head will be "turned off" by setting the weight vector to $w = (0, 1, 1)^T$. This configuration eliminates the influence of the accuracy loss signal and solely considers the other two components for calculating L_{SDQL}.

Experiment 4 is similar to experiment 1 but uses this altered weight vector w that eliminates any influence of the accuracy Q-head on the training of the model. Theoretically, this approach should help to shift the optimization procedure more towards novelty and diversity, while possibly sacrificing some accuracy. In their analysis the authors state that *"including the accuracy objective comes at the cost of diversity and novelty, while combined optimization towards diversity and novelty produce the best results with respect to these metrics"* [29]. The latter represents the setting of $w = (0, 1, 1)^T$. However, when comparing the two experiments,

which only differ in the setting of w, the one targeting only novelty and diversity performs similar but slightly worse.

This gets even more apparent in the following experiment, that additionally boosts the influence of the diversity and novelty heads by a factor of 5, as realized by setting $\alpha = 5$. Before we look at the results of this experiment, it is important to understand why it is reasonable to amplify the RL component in this context. Figure 6.3 illustrates the loss signal L_{SDQN} yielded by the RL part of the network for all the experiments, distinguished by their underlying state definitions. As a reminder, this loss signal is scaled by α and added to the Cross Entropy loss signal from the supervised head to produce the unified loss signal L_{SMORL}, which is the basis for all gradient computations. At the onset of training, the supervised loss signal typically falls within the range of 6 to 7 and after the optimization stabilizes (in well-performing models), it settles around the value of 5. On the contrary, the RL loss signal L_{SDQN} for models utilizing the SD-1 state definition lies between 0 and 0.4. Evidently, there is a substantial discrepancy between the magnitudes of the two loss signals as L_{SDQN} is more than a magnitude smaller than the supervised loss. To address this issue, the authors of SMORL introduced the parameter α [29], which can shift this imbalance to increase the effects of the RL part. In their work the authors only include the best setting for α given their experiments and do not provide the corresponding ratio between the two loss signals for this setting. Given the resemblance between their use case and the one under investigation in this study, it seems reasonable to adopt the α settings they have mentioned, especially those specifically designed for the GRU backbone, as plausible starting points. Also, it needs to be kept in mind that the RL part is intended to serve as a regularizing component. Therefore it makes sense that its corresponding loss signal is not on the same level as the main loss signal from the supervised part. However, given the considerable difference between the two loss signals, experiment 5 proceeds to use the same settings as experiment 4 while adjusting α to 5. This modification narrows the gap between the magnitudes of L_{SDQN} and the supervised loss, granting the Q-heads a greater influence on the model parameters. This is especially interesting given that even when the model's attention was concentrated on enhancing diversity and novelty with $w = (0, 1, 1)^T$, its performance remained largely consistent with experiment 3 that employed all three Q-heads. As we can see from the results in Table 6.4, the higher influence of the RL part further decreased performance. While values for NDCG@12 stayed the same and a minor improvement for repetitiveness was measured, the other metrics declined. This is especially surprising as based on both, theoretical considerations and the arguments presented in the paper, the two amplified Q-heads for diversity and novelty should have yielded an improvement in these specific objectives. This is especially true, given that the setups of experiments 4 and 5 closely mirror the methodology outlined in the paper [29].

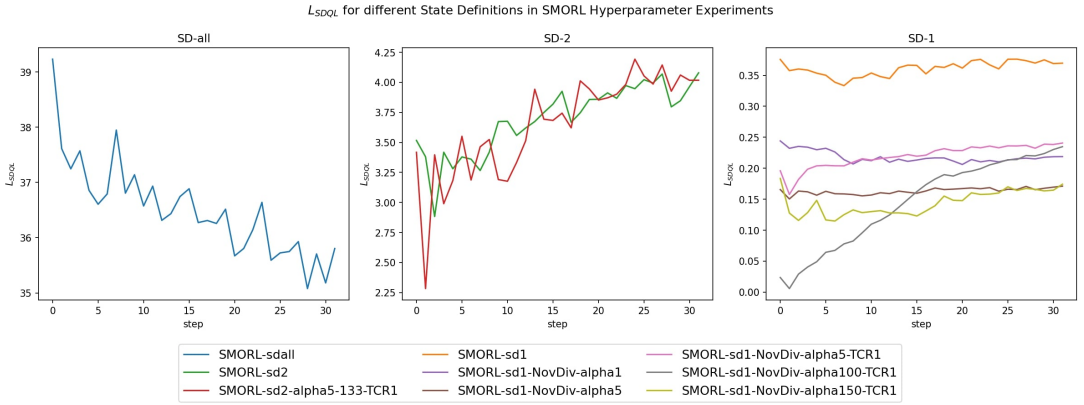


Figure 6.3: Q-Loss for SMORL hyperparameter search depending on state definition

To ensure that these unexpected results are not caused by the altered reward calculations for diversity and novelty which now incorporate the top-12 model predictions rather than just the top recommendation, the same experiment is repeated, but with TCR=1 instead. In combination with the reward structure implemented in SD-1, this creates a scenario even more closely aligned with the original experiment setup. Surprisingly, this configuration resulted in a slight decline in all measured metrics, including repetitiveness.

To investigate the effects when boosting the RL part of this setup even more, allowing L_{SDQL} to have an even greater influence on the training process, in the two subsequent experiments α is set to extreme values of 100 and 150. As expected, the observed trend continues and all metrics except for repetitiveness are drastically declining. Given the statements and results in the original paper [29], this strong boost of the novelty and diversity objective could have been expected to lead to a decline in accuracy metrics while "trading" this change for improvements in the two other objectives. However, the data from these hyperparameter experiments indicates otherwise for the given use case.

Although it is plausible that the settings in these two extreme instances were simply too high, the other prior experiments specifically targeting diversity and novelty through w in a more moderate fashion were surprising in their outcomes. As the authors state in their work, the two new Q-heads were added to *"reinforce[] diversity across a session of recommendations"* [29] and to *"introduce novelty and serendipity into the set of recommendations"* [29]. This intended behavior, however, was not evident in the presented scenarios, at least not in terms of their internal comparison within this model class. The comparison with the two other model classes, not specifically optimizing for novelty and diversity, might reveal a different, potentially positive overall effect.

One final experiment is conducted using the SD-2 state definition in an attempt to balance the described effects using $\alpha = 5$ and an objective weighting of $w = (1, 3, 3)^T$, i.e., keeping all heads active while increasing effects of both novelty and diversity. This experiment is once again executed using TCR=1, following the approach in the paper [29]. When contrasted with experiment 2, which used the same state definition but did not enhance the RL components and maintained an equal balance among the three Q-heads, this run performed considerably worse. This indicates that the boosted influence of the RL part had a negative rather than a positive effect on model performance. This outcome is again surprising, particularly as the metrics relating to the objectives that were deliberately reinforced by the Q-heads also showed a decrease.

To summarize the insights from the application of the different SMORL models on the given use case, let us begin by revisiting the Q-loss displayed in Figure 6.3. The graphs showcase L_{SDQN} of the RL part for the different state definitions, before the application of the scaling factor α. This loss represents the signal coming from the weighted linear combination of the individual Q-losses from each of the heads. To make it easier to comprehend the following discussion, the corresponding equations are once again repeated in this context:

$$L_{SMORL} = L_s + \alpha L_{SDQN} \tag{6.1}$$

$$L_{SDQN} = (L_{acc}, L_{div}, L_{nov}) \cdot w \tag{6.2}$$

Each of the individual Q-losses is computed according to the formula provided in Equation 3.4 in Chapter 3.2.4. Simplified, each individual loss for the Q-heads illustrates the discrepancy between the current state-action value that the model assigns to action A_t in state S_t for a given objective, and the bootstrapped estimate given the actual observed reward and the forecast for the maximum expected reward from S_{t+1} onwards. The ultimate aim of this Q-learning procedure in fully RL-based models is to converge towards a stable solution that provides a robust estimate of how beneficial a particular action is in a given state [30]. Ideally, as training progresses and estimates improve, this described difference should decrease.

Applied to the curves in figure 6.3 for L_{SDQN}, we can see that the model trained for SD-all has the highest loss in terms of magnitude. This can be attributed to the fact that this state

definition sums up all rewards until the next image click for the calculation of a reward in a certain state. This results in a higher reward structure in general with higher variance connected to it, as the number of interactions until the next image click as well as their type, i.e., their according reward, can be very different. This can lead to comparably high rewards for certain state action pairs in the replay buffer, which in turn causes Q-losses of higher magnitude for the accuracy head. Especially at the beginning of training, when estimates are still unstable and uncertain, and given the high variance reward structure of SD-all, this tendency is amplified. That is why L_{SDQN}, which incorporates the individual difference of Q-estimates according to Equation 6.2 for each of the three objectives, also shows high values. However, SD-all is the only model in which L_{SDQN} steadily decreases, while still being magnitudes higher than the other state definitions.

For SD-2, L_{SDQN} is of lower scale compared to SD-all, but still more than ten times higher than the one for SD-1. This can also be attributed to the fact that on the one hand, rewards are higher than for SD-1 as additionally the immediate next interaction is also included in the reward calculations. On the other hand, this leads to a slightly elevated variance in the reward structure, which might be harder to pick up for the model. However, the loss seems to be diverging for this state definition as training progresses, which is undesirable.

When comparing both other state definitions with the one shown in SD-1 models, it is observable that L_{SDQN} is in general low in scale. The reason why the values for experiment 3, highlighted in orange in Figure 6.3, differ from all other SD-1 runs is because this model also considers the accuracy Q-head, while it is excluded in the calculation of L_{SDQN} for the other SD-1 models. This state definition aligns with the one proposed in the paper [29]. It has no variance in reward structure, as only the reward for the currently clicked image in state S_t is considered, which is the same for every sample in the replay buffer. This could be the reason why there are no substantial changes in the Q-loss signal for this state definition, indicating that no fundamental learning is happening.

Summarizing these observations, particularly the different magnitudes of Q-losses in combination with their amplification through α in the computation of L_{SMORL} as depicted in Equation 3.1, the following hypothesis can be formulated: The more closely the training process mirrors the training of the basic GRU4Rec model, the better the model's performance with regard to all metrics except repetitiveness. This metric is excluded from the hypothesis as previously discussed examples have shown, that R@k can lead to misleading conclusions especially when measured on a low-accuracy model run.

When does the training procedure mirror the GRU4Rec training? The SMORL model architecture is in essence a GRU4Rec model with an additional RL part. If we were to set the value for α to zero, the model would exactly replicate the GRU4Rec model, as the RL regularization term represented by L_{SDQN} would be completely excluded from the loss computations, and consequently from the calculation of batch gradient updates. The larger the term αL_{SDQN} becomes, the more influence the Q-losses from the RL heads have on gradient computations.

When does this term increase? This term increases when either the value for α is elevated or when L_{SDQN} increases.

Given the formulated hypothesis and the setting of the conducted experiments, it is reasonable to suspect that the model settings with the highest RL impact perform worse. The highest impact, represented by a high value for the term αL_{SDQN}, is present in experiment 1 as revealed by the high magnitude of L_{SDQN} with initial values around 40. In this experiment α is set to 1. A comparable influence of the Q-heads can be found in experiments 7 and 8. While these experiments use the SD-1 state definition, leading to low-magnitude L_{SDQN}, their α values are set to 100 and 150 respectively. Given their L_{SDQN} values shown in Figure 6.3, this elevates the term αL_{SDQN} on average to comparably high levels, around 20 and above (e.g., $\sim 0.15 \cdot 150$). Table 6.6 validates these conjectures, as these experiment runs had the poorest performance.

The final experiment utilizing SD-2 with $\alpha = 5$ also yields comparably high L_{SDQN} values, which are then boosted by 5, resulting in values for the given term around 17. While being slightly better performing, it still is significantly worse than the other models with a lower magnitude.

This connection is also confirmed when comparing experiment 4 and 5. Both are using SD-1 yielding low magnitude values for L_{SDQN}. In experiment 5 these are boosted with $\alpha = 5$ while $\alpha = 1$ for experiment 4. Both models share the exact same parameters and only differ in terms of their values for α and in the random seed used to train them. Consistently, in this direct comparison, the model allowing a higher influence of the RL component performs slightly worse.

The model that most closely mimics the training of the standard GRU4Rec model is the one from experiment 3, as it demonstrates the lowest magnitude values for the term αL_{SDQN}. Utilizing SD-1 without any boosting, i.e., $\alpha = 1$, the range for this term sits around 0.35. This suggests only minor influence of the RL part on gradient updates given that the supervised loss is dominant with values around 5. Indeed, this model proved to perform best with regard to all metrics except repetitiveness. For this reason, these model settings will be selected for comparison with GRU4Rec and SQN.

Of course, the above conjecture is a simplification and merely represents a performance tendency for the models within the narrow, defined hyperparameter search. The magnitude of the loss is not the only factor that influences gradient computations and is used here more as a conceptual tool to illustrate the observed trend. As evidenced by Figure 6.3, experiment 4, with its L_{SDQN} values of 0.25, results in an even slightly lower magnitude of the RL loss term, while still yielding worse metric values compared to the best-performing experiment 3. As the hyperparameter space is vast and many other hyperparameters are kept fixed, no general statement can be made. Nevertheless, this tendency should be kept in mind for the analysis of the comparison of all three model classes GRU4Rec, SQN and SMORL, as well as for the investigation of the first two research questions underlying this work.

6.2.6 Quantitative Comparison of the Model Classes

In the hyperparameter search for each of the three models - GRU4Rec, SQN and SMORL - one candidate parameter setting for each class was chosen, that will be used for the subsequent comparison. For each of the models, four separate runs are performed, each starting with a different random initialization of model weights. This approach allows for an examination of the impact of randomness on model performance and offers insights into the model's robustness. If different model runs displayed significant variations in validation metrics, while being trained and evaluated on the same datasets, using the same hyperparameter setup, this would show, that training is not robust and overly dependent on the initial starting values of the model weights that will be optimized during training - an outcome generally considered undesirable.

Ideally, more than four models should be trained to capture the described behavior more accurately. However, as training time and resources are limited, the analysis of these four runs is used as an approximation.

Table 6.7: Hyperparameter setup for comparison experiments

Nr.	Name	Learning Rate	Packed Sequence	State Definition	w	α	TCR
1	GRU4Rec-Test	0.001	True	-	-	-	-
2	SQN-Test	0.001	True	SD-1	-	-	-
3	SMORL-Test	0.001	True	SD-1	[1,1,1]	1	12

An overview of the chosen parameter setup for the different model classes can be found in Table 6.7.

Moreover, the plots in Figure 6.4 illustrate the progression of the underlying most important metrics based on the validation set. Here, Early Stopping of the training process is used again in each of the runs, to extract the best performing model with regard to NDCG@12 given the validation set, which will then be used to evaluate on the test set. The figure only shows the best out of the four test runs for each model class, to keep a good overview.

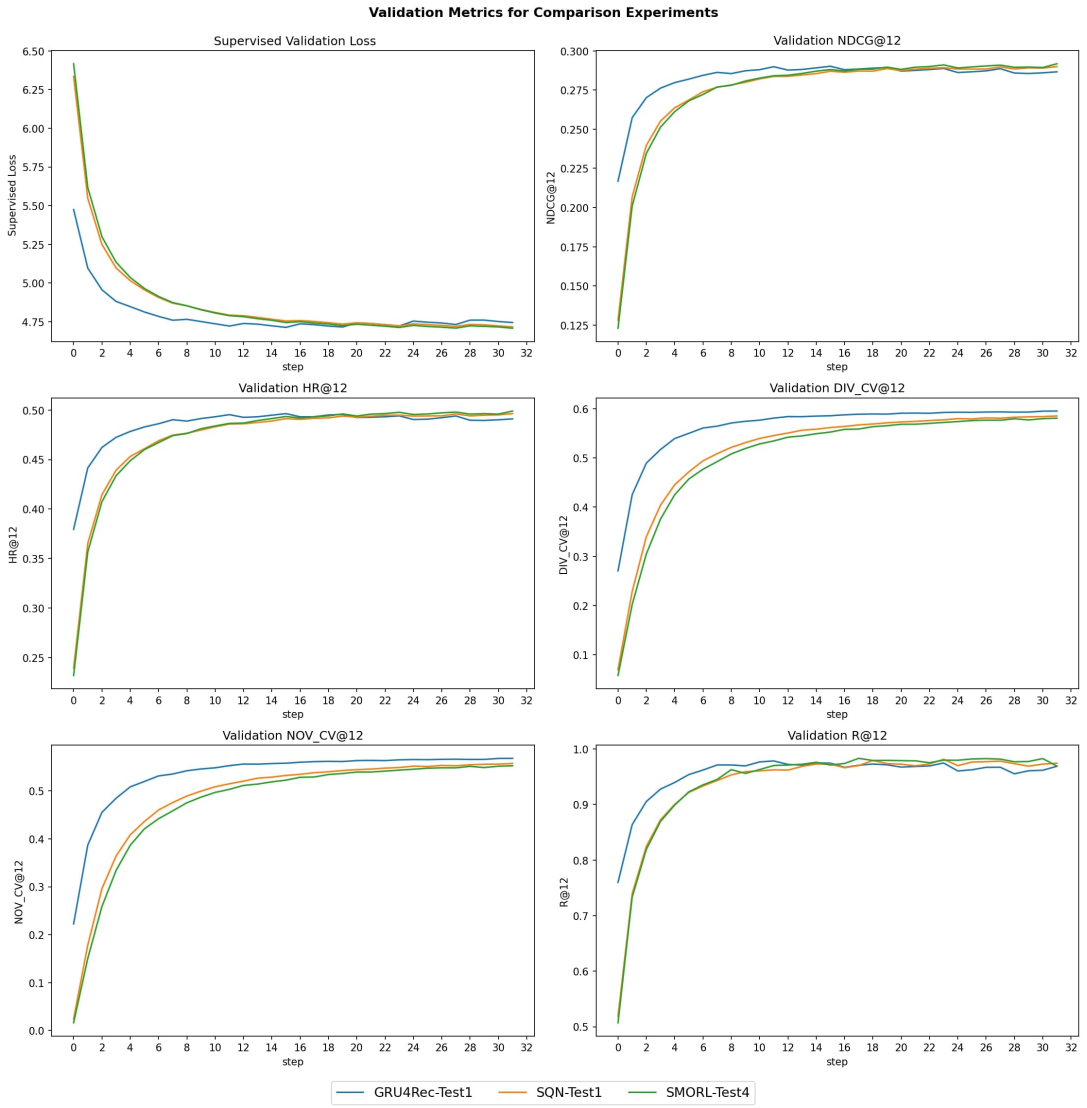


Figure 6.4: Validation metric curves for the experiments comparing GRU4Rec, SQN and SMORL. Visualized is the best performing test run for each model class with regard to NDCG@12.

As a general observation we can see that all three models are converging towards a stable version in all shown dimensions. In addition, we can see that the progress looks very similar among all models and all of them seem to converge towards quite similar results for the metrics. SQN and SMORL converge slower. This could be attributed to the fact that, in comparison to GRU4Rec, both models include a regularizing RL term in their loss computations.

Table 6.8: Test results for model comparison (μ of all four experiments per model class).

Nr.	Name	Loss	HR@12	NDCG@12	CV_{div}@12	CV_{nov}@12	R@12
1	GRU4Rec-Test	4.7188	0.4962	0.2897	**0.5842**	**0.5567**	**0.9719**
2	SQN-Test	4.7245	0.4955	0.2893	0.5828	0.5551	0.9748
3	SMORL-Test	**4.7138**	**0.4973**	**0.2907**	0.5781	0.5501	0.9731

Note: Bold denotes best value per column.

Table 6.9: Standard deviation based on the four runs per model class

Nr.	Name	σ(Loss)	σ(HR@12)	σ(NDCG@12)	$\sigma(CV_{div}@12)$	$\sigma(CV_{nov}@12)$	σ(R@12)
1	GRU4Rec-Test	0.004	0.0006	0.0002	0.0035	0.0038	0.0015
2	SQN-Test	0.0016	0.0002	0.0002	0.0005	0.0006	0.0016
3	SMORL-Test	0.0028	0.0008	0.0003	0.0027	0.0028	0.0044

The final results for the model experiments can be found in Table 6.8. For every model class, the mean of the four different runs is presented. An overview of the full result table based on the test set for each of the runs is shown in tables A.7, A.8 and A.9 in the Appendix.

In general we can observe that the mean of the three model classes is close to each other for all the reported metrics. Additionally, Table 6.9 shows the standard deviation for all metrics with regard to the four samples obtained from the different runs for each of the model classes. Overall, it is consistently low for all the dimensions which indicates that model training is stable and there are only minor effects due to randomness. Randomness in this context means differences in model results due to random changes in the training process. These random changes can arise from model weights (for example the vector representation of products and images in the embedding layer of the GRU backbone), which are initialized randomly according to the current random seed of the corresponding framework. This random state also influences the order, in which training instances are sampled from the replay buffer. As these training batches are the basis for gradient computations and therefore weight updates, this can also significantly impact the training procedure of a model. However, as discussed, these effects are only minor for the given use case, indicating generally robust training. One reason for this could be the size of the used dataset which is comparably large with over 7 million samples. This overall leads to more stable training processes, which is reflected by the consistently low variance for the four training runs in each of the model classes and for each computed metric.

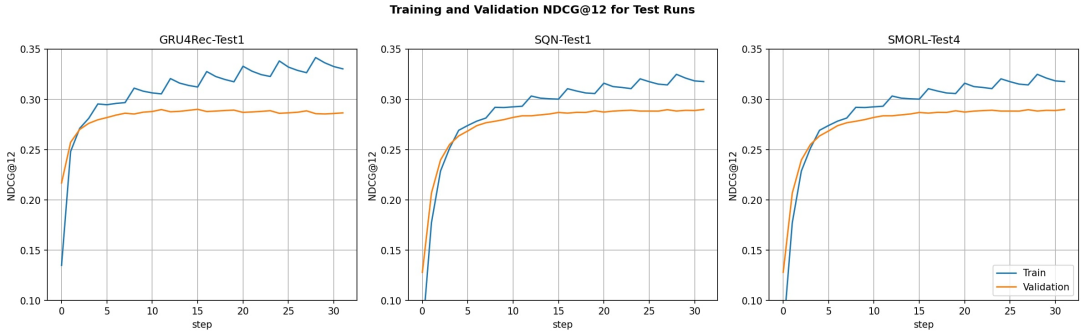


Figure 6.5: Training and validation NDCG@12 for best test runs of GRU4Rec, SQN, SMORL

Figure 6.5 illustrates the NDCG@12 accuracy metric for the three model classes based on the runs included in Figure 6.4. In the direct comparison of training and validation metrics,

we can observe that after one epoch (equalling 4 steps in the plot) all three models start to overfit the training data. While the metric is still increasing for the evaluation on the training set, the according one for the validation set almost stagnates with only minor improvements over time. The onset of overfitting occurs faster for GRU4Rec, which is consistent with the observations of a faster convergence of this model class. This underlines the potential for further improvement of the trained models with the help of other regularization techniques as dropout [28], since overfitting in general is an undesirable training characteristic.

Another observation that is striking is the oscillating behavior of the training curve with a repetitive pattern, whose size is exactly one epoch, i.e., four steps. We can see that it starts when the metric almost converged and when overfitting sets in. Likely, this phenomenon is associated with gradient instabilities or other complications in batch updates that reappear with each epoch. While other potential issues, such as non-randomized feeding of data samples from the replay buffer have been ruled out, it is clear that this effect is particularly pronounced in the GRU4Rec model and less so in the other two model classes. The reason for this is most likely again that they incorporate the RL regularization terms which contributes to gradient updates and disrupts the vanilla GRU behavior. To understand this pattern further, close monitoring of the gradients by tracking their euclidean norm is recommended, to identify any potential instabilities. Implementing mitigation strategies, such as gradient clipping, a decaying learning rate schedule, or other regularization techniques like dropout, could help alleviate this behavior.

Now the final results of the model comparison in table 6.8 are investigated in more detail. The bold values highlight the best value for the respective metric in that column. We can see that the SMORL-model on average performs best among the three classes with regard to accuracy. It has the lowest test Cross Entropy loss and the highest values for HR@12 and NDCG@12. The GRU4Rec model on the other hand dominates in the dimensions of novelty, diversity and repetitiveness.

As mentioned before, the described differences are in general very subtle. Let us take a look at the hit ratio, as this metric is straightforward to interpret. The SMORL model shows the best results with an average HR@12 over the four runs of 0.4973. That means that the real IF image click of the user in the dataset is included in the model's top-12 recommendations in 49.73% of the samples in the test set. For the GRU4Rec model, this is only the case in 49.62% of the cases. This difference applied to the amount of around 730 thousand samples in the test set, would account for approximately 800 more cases in which the SMORL model would include the real customer click in its top-12 predictions.

However, as we can see from the table, this comes at a cost: The GRU4Rec model performs better for the non-accuracy based objectives compared to SMORL. Looking at CV_{div}@12, which is indicating the proportion of the total possible next IF images that the model covers through its top-12 recommendations for the test set, we can see, that the GRU4Rec model manages to cover 58.42% of all the images. SMORL on the other hand only includes 57.81% of the images in its recommendations for the test samples. That means that in the predictions for the test instances GRU4Rec showed 61 additional unique images compared to SMORL, as the set of all possible IF images contains 10,107 IF pictures in total. A similar pattern can be seen for novelty as well. This fact is very surprising, as the SMORL model compared to GRU4Rec includes a complex RL part with multiple Q-heads, that specifically target these two objectives to reinforce novel and diverse recommendations. This is why it is counterintuitive that the opposite happened and the GRU4Rec model, while performing worse in terms of accuracy, shows better results for novelty and diversity, especially given the fact that this does not only represent the results for one model run but rather captures the average tendency across all four runs.

Of course, as mentioned before, the discussed differences in model performance are minor and overall all three models performed approximately similar. However, as SQN as well as SMORL incorporate a complex optimization procedure including RL, this fact alone is surprising. Even though SMORL is specifically optimizing for novelty and diversity, it shows the

worst results for these two dimensions. One could argue that as the model used for testing in each separate training run is taken from the step with the highest validation NDCG@12, when basing this Early Stopping decision on diversity instead, SMORL could potentially show better results to GRU4Rec, while potentially sacrificing accuracy. However, when inspecting the plot for CV_{div}@12 across all three model classes in Figure 6.4, it becomes clear that GRU4Rec consistently dominates this metric over all steps. This is equally true for novelty. Thus, both SQN and SMORL display inferior performance regarding novelty and diversity, with SMORL lagging behind the other two models across all steps, despite its unique emphasis on these aspects in its optimization procedure.

A general trend observed is that the curves representing the metrics for GRU4Rec almost represent an approximate upper boundary that the other two models converge to as well. During the hyperparameter runs of SQN and SMORL, the parameter settings were advantageous for both models, which made the training process resemble the one of GRU4Rec more closely. For SQN, the state definition SD-1 with the lowest variance in reward structure proved to be superior with regard to all metrics except repetitiveness. This is a lot easier to pick up for the model and results in generally lower values for the RL regularization term in the loss, reducing its influence on weight updates, therefore resembling the GRU4Rec training more closely. The same connection could be observed for the SMORL model as in this context SD-1 also proved to be the best choice given the conducted experiments.

Implications for RQ1

The comparative analysis presented in this chapter provides insights into the first research question. The purpose of RQ1 is to investigate, whether for the given use case of IKEA's IF, incorporating a RL part into the supervised network architecture would provide benefits for the accuracy of the model, measured by the two metrics HR and NDCG for next image recommendation. Specifically, this question targets the comparison of the purely supervised baseline approach GRU4Rec with the SQN model.

In the original paper, the authors state that *"the proposed SQN method achieves consistently better performance than the corresponding baseline"* [39]. The GRU4Rec model is one such referenced baseline, and the SQN model with a GRU backbone is the model that outperformed this baseline in their experiments.

The comparison used for this work closely follows their experiment setup including the specific parameter configurations of the model. Given the results for this comparative analysis, it is evident that RQ1 cannot be answered positively. The SQN model demonstrated reduced performance in every assessed dimension, including accuracy, when contrasted with the baseline GRU4Rec, as shown in Table 6.8. Naturally, as discussed before, these results are based on the limited hyperparameter search conducted earlier. A broader search may have possibly yielded different results. For the tested approach however, the RL component did not help the model to learn weights that are advantageous for next image recommendation. Instead, it appears to have disrupted the learning process, diminishing the model's predictive power and consequently its accuracy in correctly predicting the next customer image click based on the previous interaction window of size ten.

Moreover, it needs to be mentioned that the model performed relatively worse, but changes were minor in general. This can be attributed to the fact, that due to the used state definition, which was proven to be beneficial in the hyperparameter search, the RL part has only a minor influence on model training. Other tested parameter settings on the other hand, with a higher influence of the RL-regularizer, yielded significantly worse results when evaluated on the validation set (e.g., experiment 1 using SD-all with over 50% decrease in HR@12 compared to experiment 5).

Implications for RQ2

The second research question serves to investigate whether adding two additional Q-heads to the SQN architecture, representing a form of multi-objective optimization procedure, will improve all or at least some of the investigated metrics for accuracy, novelty, diversity and repetitiveness. Essentially, this question concerns the comparison of the GRU4Rec and the newly introduced SMORL model, given that SQN did not demonstrate any advantage in the overall model comparison for the use case at hand.

The comparison procedure, including the calculated metrics and choices for model parameters, followed the approach presented in the original SMORL paper [29]. Their experimental results showed "*a substantial increase in aggregate diversity, a moderate increase in accuracy, reduced repetitiveness of recommendations*" [29]. Based on the two example datasets, their results suggest that the SMORL model improved all investigated metrics for all included objectives. For example CV_{div}@10 rose by over 7% and CV_{nov}@10 increased over 9%.

For the conducted comparative analysis in general, SMORL performed on average very close to GRU4Rec with regard to the provided metrics in Table 6.8. The results for accuracy, represented by HR@12 and NDCG@12, showed an improvement compared to the single-objective Supervised Learning baseline, whereas performance decreased for the three other metrics related to diversity, novelty and repetitiveness. As discussed before, this is particularly surprising, as the model specifically designed to enhance these objectives, performed worse in the conducted analysis.

Therefore, for RQ2, the incorporation of the two additional Q-heads targeting novelty and diversity generally led to an increase in recommendation accuracy while degrading other metrics. Considering that the metric changes are overall minor and the model introduces additional complexity concerning training and deployment, it does not seem practical to select SMORL over GRU4Rec for the given use case, based on the investigated model settings. It is important to note again, however, that the hyperparameter search was limited, and a broader search could yield different results.

Finally, it is highly questionable whether an approach should be adopted that specifically optimizes for two goals that did not improve but rather deteriorate compared to a simpler single-objective baseline.

6.2.7 Qualitative Comparison of the Model Classes

In the following, an additional qualitative analysis of some example recommendations for each of the three model classes will be investigated. It is important to underscore that this analysis lacks complete objectivity, with its sole purpose being a visual evaluation of the recommendation behavior.

To accomplish this objective, samples from the test set are taken, recommendations are retrieved for GRU4Rec, SQN and SMORL, and are then visualized. Figure 6.6, 6.7 and 6.8 showcase six different examples for this. The first row illustrates the real recorded customer sequence of size 10, representing the interactions in the last ten timesteps, that is used as an input for the models. The image surrounded by a green frame is the IF image that the customer chose to click on next. The three following rows of pictures illustrate the top-11 recommendations, yielded by each individual model. The percentage over each image states the according probability that the model assigns this specific picture. This is calculated by feeding the output logits for each of the images, yielded by the last layer of the network, to the Softmax function, that normalizes this vector of logits to a probability distribution over all approximately 10 thousand IF images. The images are sorted in ascending order, with the most probable one on the right. To produce these results, for each model class the best performing model from the four test runs is used. Only examples are shown where at least one of the models included the real image click in the top-11 recommendations.

In general, the recommendations appear plausible when considering the featured products and thematic context relative to the input sequence. Additionally, the top-11 predictions from each model class are broadly similar, which is expected given the comparable performance on the test data across various metrics. Nevertheless, despite the closely aligned accuracy values for HR@12 and NDCG@12, it is interesting to see the differences in the ordering and selection of images among the models. This demonstrates that even modifications to the model, reflected only in a subtle change in test-metrics, can result in different recommendation behaviors. Ultimately, this underscores that the training process is intrinsically a stochastic procedure and small variations can lead to different outcomes.

In some cases, all models agree in their top prediction which is showcased in Figure 6.6b or Figure 6.8a. In the latter, it seems to be an easy decision for all three models - the actual next clicked image is the most likely recommendation for GRU4Rec, SQN, and SMORL, each assigning it probabilities exceeding 30%. For most examples on the other hand, models do not agree and show different results. In some situations, one model might do better than the others. But in general there is no way to objectively evaluate which recommendation is truly the best in the given situation. This is strongly connected to the individual customers' preferences, which are obviously unknown.

An interesting example showing that the models are not only focused on the most recent interaction is illustrated in 6.7a. The customer first viewed a couple of kitchen accessories like different food storage boxes and also the storage turntable "SNURRAD"[2], which was presented in the embedding analysis already. At timestep t, they viewed black hangers after which clicking on the IF showing a clothes rail. It is interesting to note that all three models included the actual image the customer clicked on in their top two suggestions. However, the other suggested images were still mostly related to the kitchen theme. This is reasonable as the majority of the customer's browsing history was focused on kitchen items. This shows that even in a situation in which two entirely different categories of products are present in the historic sequence, the model can still consider the most recent interaction but also remember the theme of earlier actions when making suggestions.

A common problem of RSs is the popularity bias [1]. It describes the phenomenon, that very popular items are in general recommended very frequently, while other items are only shown occasionally. This bias can be observed in the provided examples as well. The left most IF of the SQN predictions in figure 6.7a is the most popular image on the IKEA website for the 41-day period covered by the datasets. It features a small living room table with built-in speakers. Given the customer's previous interest in kitchen storage accessories and hangers, this does not seem like a relevant suggestion. We can see this same image appearing in several other top-11 suggestions, like in Figure 6.6b for GRU4Rec, or in Figure 6.7b for both SQN and SMORL. In none of these cases does it seem particularly suitable, so it ends up acting more like background noise than a valuable recommendation. This can also be seen when investigating some of the additional examples included in the Appendix in Figure A.6, A.7, and A.8.

The two scenarios depicted in Figure 6.7 represent common situations in which a model would have to generate recommendations based on a customer's recent interactions on the website. Specifically, these sequences comprise only product views followed by a single IF click. As we discussed in Chapter 4, more than half of all recorded sessions consist of just one IF interaction, which mirrors the situations in the referenced examples.

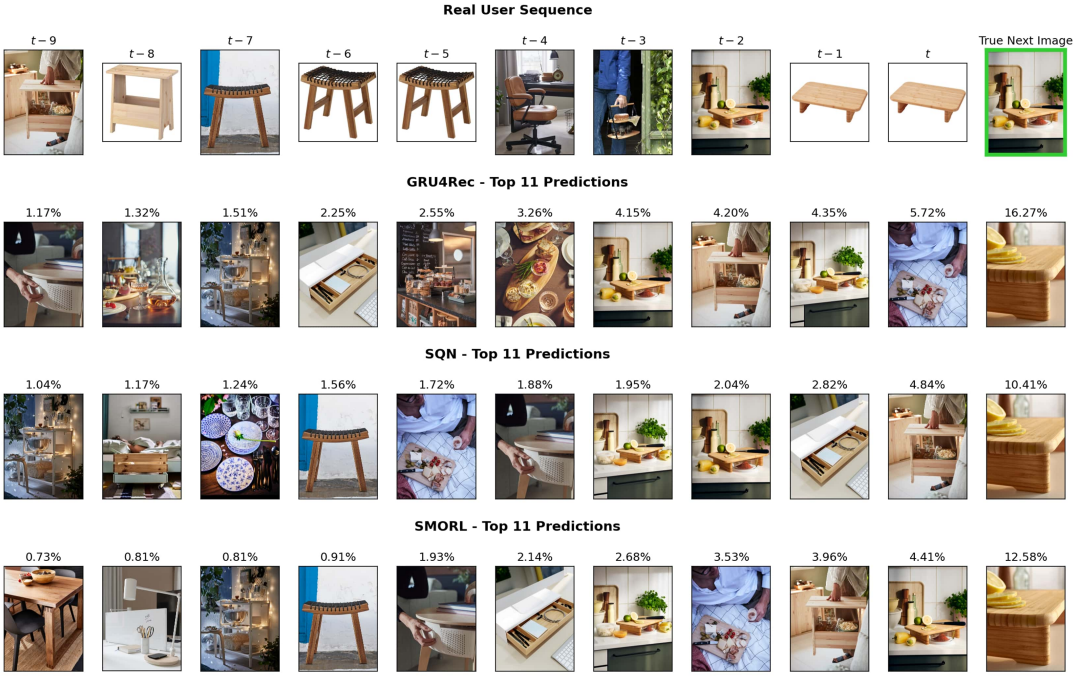
The second scenario in Figure 6.7b highlights an additional characteristic of the input sequences: repeated views of the same product in different colors. This happens when customers switch the color of the product they are viewing, which registers as distinct interactions. These repeated views might introduce noise as they potentially represent redundant information. As we noted earlier, the model will learn similar vector representations for the

[2] https://www.ikea.com/se/sv/p/snurrad-snurrbricka-foer-foervaring-transparent-90506104/

same product (even in different colors) during training. It remains unclear if viewing an item in multiple colors offers any valuable information for predicting the next IF image to display.

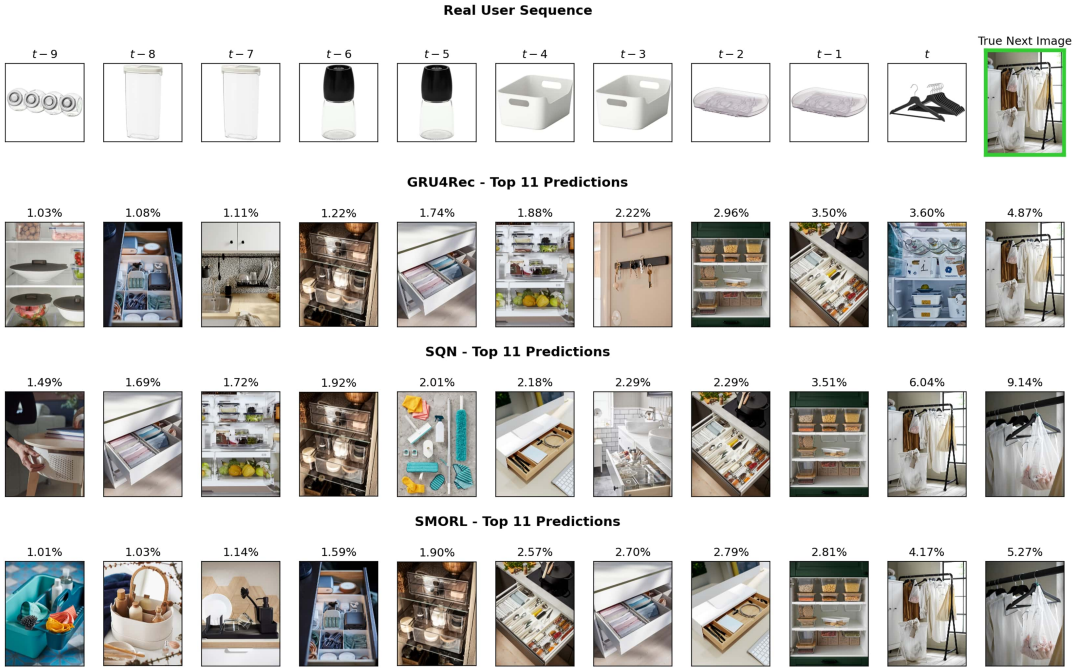
Nevertheless, in the same example we can observe that while GRU4Rec's and SMORL's recommendations are strongly focused on the most recent view of a shower faucet, SQN also includes IF images in the context of armchairs.

(a) First example recommendations



(b) Second example recommendations

Figure 6.6: Comparison of recommendations by GRU4Rec, SQN and SMORL (1)



(a) Third example recommendations

(b) Fourth example recommendations

Figure 6.7: Comparison of recommendations by GRU4Rec, SQN and SMORL (2)

(a) Fifth example recommendations

(b) Sixth example recommendations

Figure 6.8: Comparison of recommendations by GRU4Rec, SQN and SMORL (3)

6.3 Window Size for Prior Interaction History

This section studies the influence of the number of prior interactions in the input sequence on model performance and recommendation characteristics, aiming to detect possible long-term dependencies in the data. After an initial quantitative analysis, a qualitative assessment of two selected examples is conducted.

6.3.1 Quantitative Analysis

We select the GRU4Rec model with the hyperparameters that performed best during the hyperparameter search, training it similar to the models analyzed earlier. This involves training on samples from the training set, applying Early Stopping based on the validation NDCG@12, and evaluating the model using the test set. The only difference in the experiments is that the length of the input sequence will be varied. Investigated window sizes are 1, 2, 3, 5, 10 and 20 as shown in Table 6.10. As a reminder, all previous models used a fixed sequence length of 10 according to the specifications in the paper [29].

Table 6.10: Experiments for different window sizes

Nr.	Name	Window Size
1	GRU4Rec-s1	1
2	GRU4Rec-s2	2
3	GRU4Rec-s3	3
4	GRU4Rec-s5	5
5	GRU4Rec-s10	10
6	GRU4Rec-s20	20

Table 6.11 displays the results of the conducted experiments based on the test dataset. For these experiments, the repetitiveness metric is excluded. As discussed in Chapter 5.4.4, this metric is dependent on the state size, as it measures how often on average an IF image click that is already present in the prior interaction sequence of the customer is repeated in the top-k recommendations. Since this is more likely to occur for larger window sizes, it is unfair to compare when window size is varied. Therefore it is excluded from the analysis in this context.

Table 6.11: Test results for window size experiments

Nr.	Name	Loss	HR@12	NDCG@12	CV_{div}@12	CV_{nov}@12
0	GRU4Rec-s1	4.9107	0.4815	0.2828	0.5894	0.5622
1	GRU4Rec-s2	4.8220	0.4870	0.2855	0.5917	0.5646
2	GRU4Rec-s3	4.7749	0.4903	0.2867	0.5863	0.5589
3	GRU4Rec-s5	4.7397	0.4939	0.2883	0.5859	0.5585
4	GRU4Rec-s10	4.7181	0.4952	0.2891	0.5859	0.5585
5	GRU4Rec-s20	4.7039	0.4985	0.2909	0.5852	0.5577

The results show that the window size of historical interactions has a positive correlation with the Cross-Entropy loss and both accuracy metrics, affirming that a larger window size supplies the model with more information, improving its ability to accurately predict the next clicked image.

However, the changes are surprisingly moderate. The model trained on a window size of 20 contains the actual next image clicked by the customer in the top-12 recommendations 49.85% of the times. The model that only uses the most frequent product or image interacted with by the user still reaches a rate of 48.15%. The fact that only having access to a single last interaction does not drastically worsen the quality of recommendations is unexpected. This effectively means that each user that clicked on a specific product in the last time step would consequently be recommended the exact same set of top-12 recommendations, as no other information except this last step are incorporated to calculate the predictions. Let us investigate some real prediction batches to get a deeper understanding of the real recommendation behavior, beyond the metrics.

6.3.2 Qualitative Analysis

In this qualitative analysis, the recommendations for each of the different models are analyzed. In the following first example, the user interaction sequence from 6.7a is reused. It contains diverse product clicks, for kitchen storage accessories as well as clothes hangers, which makes it a good candidate to study the effect of varied sequence lengths. The prediction results together with the input sequence for each model are depicted in Figure 6.9. The top row shows the input sequence provided to the model.

We can see, that each of the models includes the true next image click in its top-11 illustrated predictions. All models, with the exception of the one employing a window size of five, even predict it as their top recommendation. This implies that when just evaluating the yielded recommendations based on accuracy, for each model HR@11 and NDCG@11 would be 1[3]. However, when analyzing the full set of recommendations, significant differences can be observed.

Consider the model in Figure 6.9a that only uses one prior interaction. It receives the product-click on the black hangers as an input. The subsequent predictions mostly contain IF images showcasing clothing storage options, with most of them incorporating the input hangers. There is also one picture showing stored foods, which is ranked lowest.

When the model has access to the next prior timestep as well, which refers to the storage turntable "SNURRAD", recommendations shift: As depicted in Figure 6.9b, the IF image with the stored foods has now rank three. Also, the set of other pictures shown in the top-11 predictions changed as well.

As a third timestep is incorporated, presenting an additional click on "SNURRAD", the recommendations undergo a significant transformation. This is illustrated in Figure 6.9c. While the overall theme of the top-11 predictions for the model utilizing a window size of 2 was primary clothing-oriented, it shifted when the model has access to the last three interactions. Here, the true next picture of the clothing rack is still the top recommendation, but all other predictions are mostly food storage related.

This pattern mostly stays the same for larger window sizes, as shown in Figure 6.9d and 6.9e. The corresponding visualization for the model using a window size of 20 is included in the Appendix in Figure A.9.

While in this specific example the most frequent interaction might be enough to correctly predict the next IF image click, the overall set of recommendations differs significantly. When only one timestep is used as input, the model is limited to the available information. Consequently the predictions are focused on the theme included in this input. When more timesteps are taken into account, more diverse sets of recommendations are yielded by the model. It not solely focuses on the most frequent interaction, but also captures other patterns from previous clicks, thereby detecting additional areas of interest for the customer. Which recommendations truly represent a more interesting experience for the customer is not definitively

[3]For the model using window size of 5, NDCG@11 is smaller than 1, as the target image is not the top-ranked recommendation.

apparent. Nonetheless, it appears plausible that customers might generally favor the more diverse feed produced by models utilizing larger window sizes, as the recommendations appear more personalized.

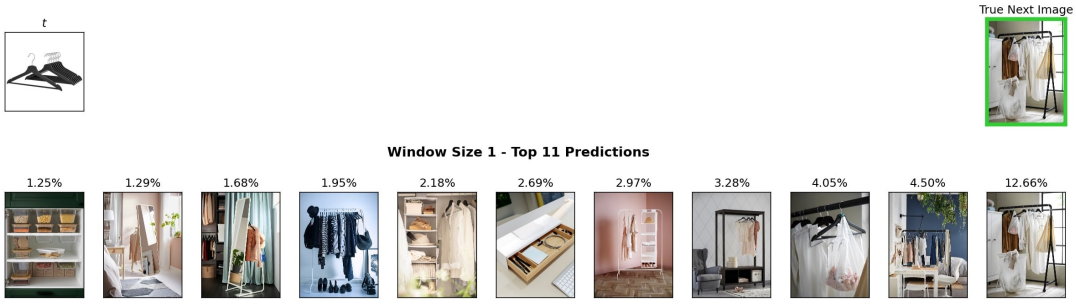
The second example showcases another way of analyzing the influence of the window size on resulting recommendations. Instead of using an existing sample from the test set, the interaction history is artificially designed. For the scenario illustrated in Figure 6.10, a combination of frequently clicked IKEA products is chosen. In general, this approach can be used to simulate the impact of combining different products into the window of prior interactions on top-recommendation batches. The designed click-stream for the following example is comprised of a quadratic white shelf, another brown wooden shelf and the glass food storage box. For simplicity, in this context only models up to a window size of 5 were used.

We can see in Figure 6.10a that when only the glass box is used as input, the recommendations are very focused on IF images related to food storage options.

The addition of the brown shelf as another prior interaction introduces some variety. While top recommendations continue to exhibit refrigerators with food boxes, other images now incorporate this specific brown shelf. This is illustrated in figures 6.10b and 6.10c.

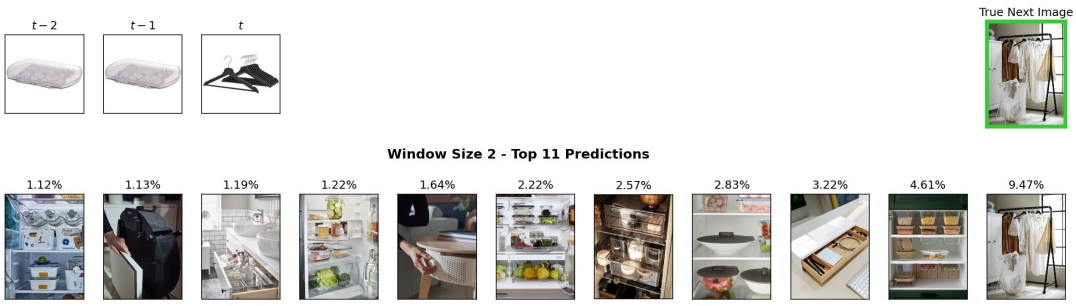
To test the model utilizing a window size of 5, two more interactions with the white shelf are added to the artificial user history. The corresponding visualization is included in Figure 6.10d. It is interesting to see, that in this situation the presented recommendations include IF images connected to each of the three products, i.e. the customer's three possible areas of interest. From a qualitative perspective, this recommendation feed appears substantially more diverse than when a single timestep is used for prediction.

In summary, the qualitative analysis demonstrates that even with metrics displaying similar values in the quantitative analysis, the actual recommendations can significantly differ. All models successfully include the true next image in their top-11 predictions, typically in a highly ranked position. This forms the basis for the calculation of the accuracy metrics. The additional recommendations yielded alongside the true next image do not influence HR or NDCG. The authors of the SMORL paper attempt to capture this behavior with their introduced diversity and novelty metrics [29]. However, as the experiments for the window size revealed, it is questionable if these metrics truly reflect the underlying diversity and novelty of recommendations. Even though, for the two shown examples, models incorporating a larger number of prior customer interactions seem to yield a more diverse feed of predictions, the diverstiy metric even decreases slightly for larger window sizes. This discrepancy further highlights that the offline metrics proposed by Stamenkovic et al. [29], which are used in the conducted experiments for this work, do not manage to capture the models' true recommendation behavior, at least in the given scenario. They do not offer enough insight in this setting and are therefore insufficient for a thorough comparison of different models.

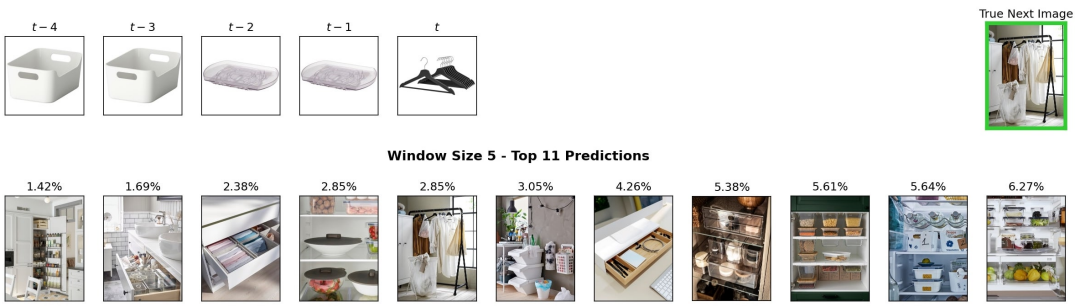


(a) GRU4Rec-s1 with window size 1

(b) GRU4Rec-s2 with window size 2



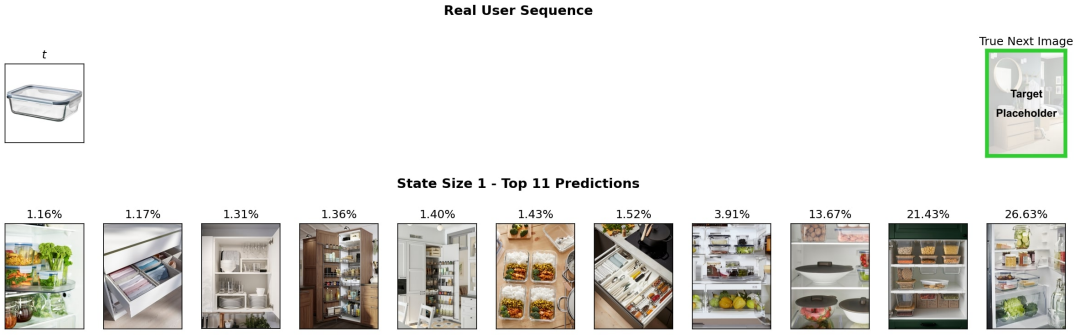
(c) GRU4Rec-s3 with window size 3



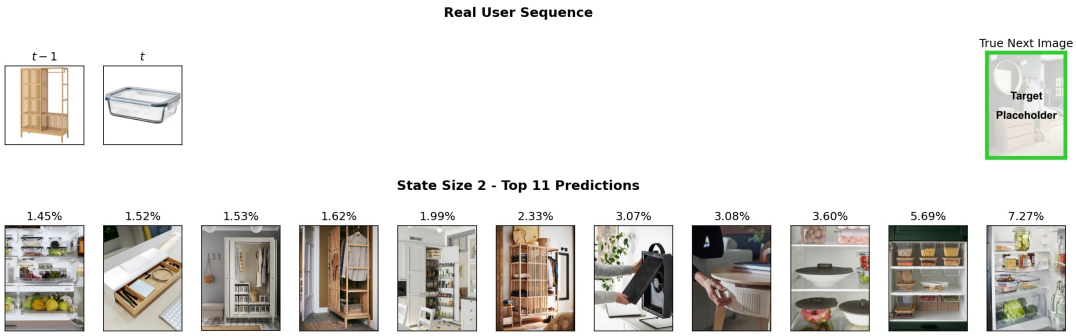
(d) GRU4Rec-s5 with window size 5

(e) GRU4Rec-s10 with window size 10

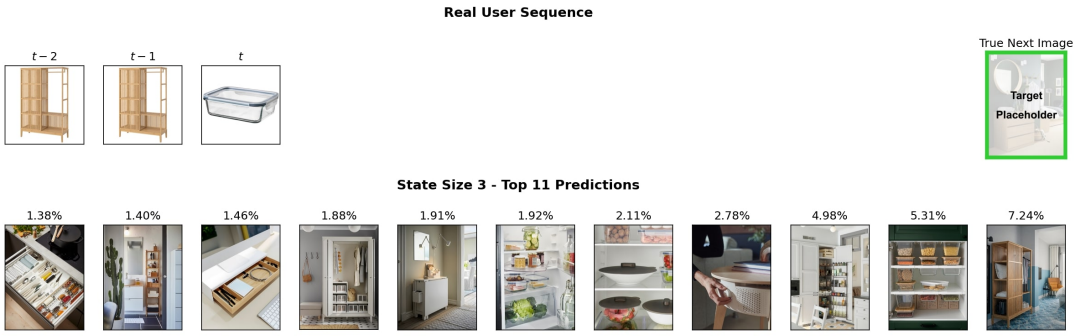
Figure 6.9: Example recommendations for different window sizes for GRU4Rec models



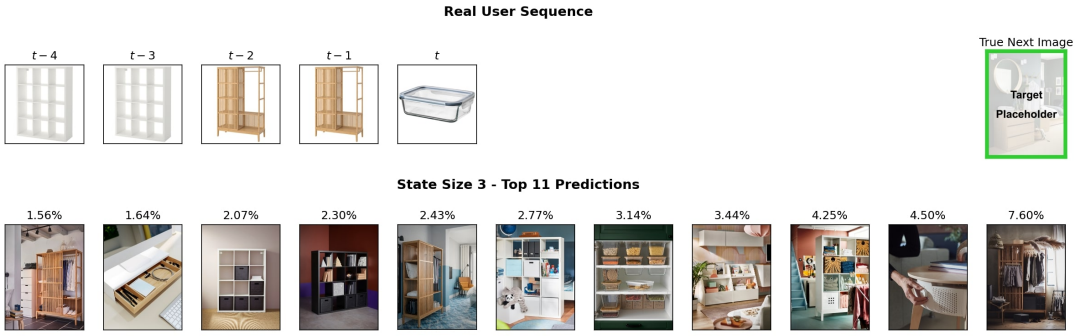
(a) GRU4Rec-s1 with window size 1



(b) GRU4Rec-s2 with window size 2



(c) GRU4Rec-s3 with window size 3



(d) GRU4Rec-s5 with window size 5

Figure 6.10: Example recommendations for artificially created click sequence based on window size experiment models

7 Discussion

This chapter discusses the limitations of various aspects of the research conducted in this work and explores potential directions for future research. Covered areas include the used data, the chosen models and the evaluation procedure.

7.1 Data

The following section presents general limitations connected to the data underlying this work.

7.1.1 Bias of Time Horizon

The data utilized in all experiments conducted in this study spans 41 consecutive days from December 2022 to January 2023. It is plausible that distinct patterns in user behavior may emerge at different times of the year. For instance, customers may exhibit a higher interest in outdoor or balcony furniture during spring and summer than in winter. This seasonal variation will likely be reflected in the data, leading to underrepresentation of products or IF images from such categories during the model training process. Given that the trained RS should perform optimally throughout the year, it would be beneficial to use a more diverse dataset for both training and evaluation. This could be accomplished by incorporating user interaction sequences from various periods throughout the year.

Generally, future experiments in this area could benefit from utilizing a more extensive dataset. This larger dataset could be paired with a more complex, higher-capacity model with an increased number of parameters. The expansive dataset would provide the necessary stability during training, even with a more sophisticated model configuration.

7.1.2 Preprocessing

The preprocessing part for the underlying dataset in this work was limited to the exclusion of missing data and transformations to convert the dataset into the described replay buffers. There is a lot more potential in this part of the study.

As we have seen in the embedding analysis, the set of all IF images contains duplicates as well as pairs of pictures that show exactly the same room installation just from a slight

different angle or zoom. As a reminder, some examples of this can be found in A.2b in the Appendix. This issue complicates the modeling process since duplicates, despite their identical content, are treated as distinct entities by the model. Further, their corresponding vector representations in the embedding layer tend to be highly similar, as identified in Chapter 6.1, which makes repetitions more likely to occur in the model's recommendations. However, these repetitions could not be captured by the employed repetitiveness metric as the duplicates have different indexes. Future experiments could explore mapping these duplicates to a single index. But this is obviously a tedious procedure due to the need for visual analysis. These potential benefits would need to be carefully evaluated against its associated effort and costs.

The problem is even worse for the product dimension. As we saw in the qualitative analysis of model predictions in Chapter 6.2.7, it is a very common phenomenon in the data that many customer sequences contain the same product for multiple subsequent timesteps. This is a characteristic of the data collection process connected to IKEA's website, as views of different colors of the same product are registered as different product clicks. This problem should be addressed in the preprocessing pipeline, as it likely has negative influence on the training process and the overall model's performance.

On the one hand, this leads to a general overrepresentation of certain products in the dataset as some products might be available in a wider range of colors than others or are just viewed more often in different colors.

On the other hand, customer interaction sequences are inflated with these duplicates, especially as they can range over more than five timesteps. But does this add useful information for the model to learn from? While it is difficult to make a general statement, it is highly probable that this redundancy fails to provide additional useful insight. It is reasonable to assume, that the different colors of a certain product viewed in the past will have no particular influence on the IF preferences of the customer in the curent timestep. Hence, these duplicates potentially add unnecessary noise and displace older, possibly crucial, interactions from the preceding interaction window. Moreover, since each color variation of a product corresponds to a unique index, the number of model parameters is inflated too, as each color variant is represented by a 64-dimensional vector.

The practice of treating the same product in different colors as separate vectors in the embedding space poses another challenge. If a certain color variant is less popular and consequently less represented in the dataset, the model might struggle to learn a stable and accurate vector representation for that variant. This issue could lead to a disparity in the quality of recommendations: customers interacting with a popular color variant, for which a stable embedding has been learned, could receive superior recommendations compared to those interacting with a less frequent color variant. This problem would also be eliminated by mapping the same product of different colors to the same label in the data corpus.

7.1.3 Popularity Bias

As investigated in the qualitative analysis of the trained models, there is a popularity bias included in the data, which is also reflected in the model's recommendations. In Figure 5.2 the discrepancy between the long-tail images, which make up over 90% of all IF pictures, and the small amount of highly popular images is highlighted. As we can see, the frequency disparity between these two categories of images is extreme, resulting in an overrepresentation of a few selected images in the dataset. This biases the training procedure and results in the observed issue, that even if unrelated, these images will often rank among the top-12 recommendations. This problem should also be addressed, as this bias will likely lead to lower satisfaction with the RS as some popular images are repeated in many situations.

A potential strategy to mitigate this problem could involve undersampling instances in the dataset that contain these exceptionally popular images. Typically, this approach presents a challenge due to limited data availability, but in the context of this study, this concern does

not apply since the used data only represents a fraction of the full corpus. Implementing such a strategy could help to rebalance the dataset, reduce the popularity bias, and consequently improve the quality and variety of recommendations.

Additionally, it is important to focus on including a metric that can accurately measure and address the problem of popularity bias. The metrics used in this study were not able to fully capture this undesired behavior.

7.2 Models

This section focuses on discussing the choice for the models used throughout the experiments, including their implementation and the conducted hyperparameter searches.

7.2.1 Model Performance

The selection of models examined in this study was guided by a review of the literature and the specific attributes of the use case in question. The comparative analysis included three closely related models GRU4Rec, SQN and SMORL. GRU4Rec as a representative of the supervised model class, SQN with a GRU backbone as a RL extension to GRU4Rec and SMORL with the same backbone network extending the architecture of SQN with two additional Q-heads targeting diversity and novelty of recommendations. As discussed in Chapter 3.1.3, the original author's findings suggested promising results [39], [29]. In essence they argue that SMORL significantly improved performance across all reported dimensions compared to GRU4Rec and SQN [29]. However, applied to the use case in this work, this was not observed. The following sections discuss potential reasons for this.

Hyperparameter Search

The final comparison of the three model classes was based on a preceding hyperparameter search for each model type. This exploratory phase aimed at experimenting with different parameter settings to find an appropriate one to be used for a final comparison. While multiple different variations were tried, the hyperparameter space, especially for the models SQN and SMORL including RL, are extensive. Given time constraints and computational resource limitations, only a small portion of this space could be covered. The comparison of the models was based on the limited parameter search. Further experiments should be conducted, exploring more diverse settings which could potentially yield differing results.

Implementation

As mentioned before, each of the models including the corresponding evaluation framework were implemented from scratch. This poses a potential risk, as an undetected flaw in the underlying code could cause the models to not learn as intended, potentially influencing results negatively.

The authors made their official code for the original paper [29] available, which allowed for an in-depth analysis of their implementation. This code was thoroughly examined to ensure the accuracy and correctness of the developed implementation for this.

In the next phase of investigation, the authors' original code was applied to one of the datasets featured in their paper, aiming to gain a deeper understanding of the target learning behavior to be replicated for IKEA's IF. The models - GRU4Rec, SQN, and SMORL - were configured with the exact parameter settings as presented in the paper, utilizing their existing codebase on the RetailRocket dataset. This dataset was also included in the provided files by the authors and was already split in training, validation and test set. Using this exact architecture, parameter settings and their evaluation framework with all reported metrics in the paper, the models were subsequently trained on the stated train set and evaluated on the test

set. Figure 7.1 showcases the progress of the metrics for the test dataset for the three different models, GRU4Rec, SQN and SMORL. Each step corresponds to the evaluation interval, which is roughly equal to 2.5 epochs. The same learning rate of 0.005 was applied for all three models, following the explanations in the paper. While this rate appears well-suited for SQN and SMORL, GRU4Rec's performance immediately declines as evident from the HR@10 and NDCG@10 curves. This led to conducting an additional experiment for GRU4Rec, maintaining the same setup but with a reduced learning rate of 0.001 (indicated by the red dotted line). As shown in the graph, this adjustment resulted in better performance across all metrics for GRU4Rec, outperforming the model configuration from the original paper.

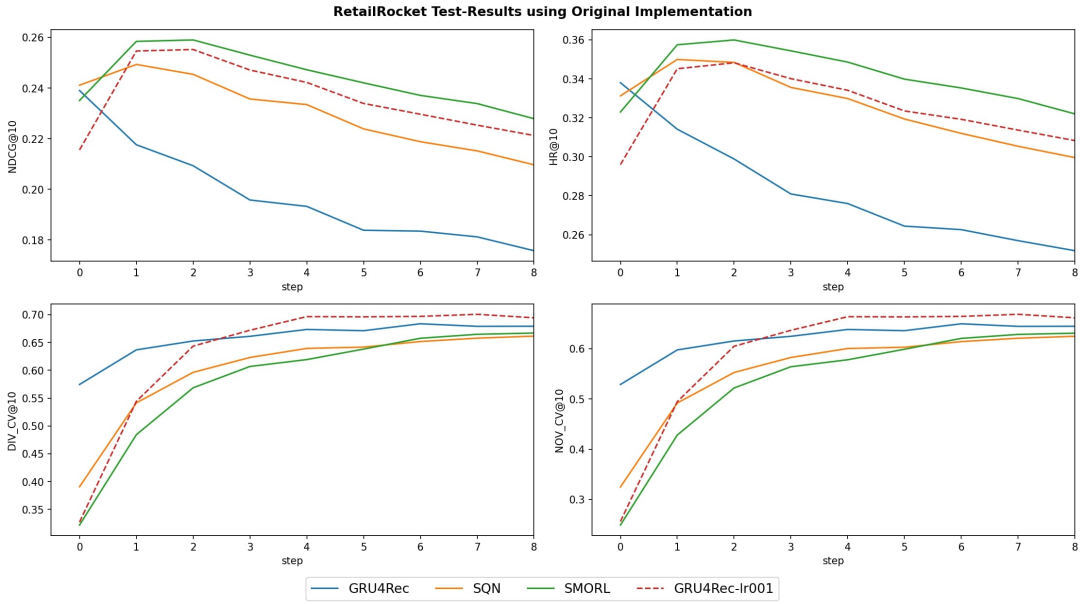


Figure 7.1: Test results for RetailRocket dataset using original paper implementation [29]

Looking at these results, we can see a pattern reminiscent of the observations made in the experiments conducted in this thesis. For accuracy, SMORL exhibits a minor improvement in NDCG@10 (0.259 vs. 0.2552, an approximately 1.5% increase). However, the additional run of GRU4Rec, with a reduced learning rate, seems to outperform both SQN and SMORL in terms of novelty and diversity, which is consistent with the observations connected to the experiments in this work. Even when comparing SQN and SMORL to the original GRU4Rec run with the disadvantageous learning rate defined in the paper, neither SQN nor SMORL show any improvement in novelty or diversity. The models appear to converge to comparable values, with GRU4Rec mildly dominating in these areas. This outcome is somewhat surprising, given the authors' claimed improvements in these newly introduced objectives (over 12% for diversity and 15% for novelty based on CV@10). This is particularly unexpected considering that SMORL explicitly includes these objectives in its optimization process. So even when using the exact code base provided by the authors in combination with their parameter setting, ignoring the fact that the chosen learning rate is not feasible for GRU4Rec as shown in the plots, on their provided test dataset with their evaluation protocol, the promising results claimed in the paper were not reproducible. While one could argue that the authors used 5-fold cross-validation to obtain their results, it seems highly improbable that this exact train-test split would yield such different outcomes. Furthermore, the employment of this evaluation procedure is also questionable, as there is no evidence of cross-validation being actually used in their provided code.

In summary, this at least raises questions regarding the reproducibility of their results and therefore also the feasibility of their models to improve novelty and diversity in recom-

mendations. Further research should be done to better understand the true effects of the RL components in SQN and SMORL on model training to investigate whether, and under what conditions they truly improve model performance with regard to the reported dimensions. This is especially true since they substantially augment complexity, not only in terms of model architecture but also in relation to the hyperparameter search and data preparation. As evidenced by the experiments in this study, the design of states and rewards can significantly influence model training, and the introduction of new sets of hyperparameters for the two models only adds to this complexity.

Against the background of these inconsistent results, the observed behavior of the models trained in this study does not appear too unexpected anymore. However, it is important to note that this does not necessarily rule out the existence of implementation errors in the code employed for this research.

Furhtermore, it must be noted that the discussed results for the paper only cover the authors experiments with a GRU-backbone. A starting point for further experiments for IKEA's IF could be the employment of other backbone models like Caser [31] or SASRec [16].

7.2.2 Model Choice

The choice for the models used throughout this work was driven by the literature review and the characteristics of the given use case.

An interesting future research approach could be to take the insights from the experiment and compare the trained models with purely RL based approaches. While this might pose an even greater challenge concerning effective offline evaluation procedures, it could be insightful to investigate in this direction. Especially the field of offline RL with algorithms like the recently proposed Conservative Q-Learning approach [18] could pose a promising starting point for this.

The literature review and subsequent model selection primarily concentrated on session-based recommendation models, which consider a sequence of previous interactions as input. However, it is worth noting that conducting a broader exploration of models may uncover additional compelling candidates for the specific use case at hand.

7.3 Evaluation

The following section investigates potential problems in the evaluation procedure used in this work.

One of the significant challenges in training RS models is that they are trained on a collection of offline customer interactions and evaluated in the same offline manner. Ideally, the most reliable method of comparing different RS would be through online testing, involving real-time user interactions. This way, the real impact of the change in the model could be measured. For example, it could be measured, how many consecutive clicks customers perform in the context of the IF, if more visits to the IF can be observed, if customers are more likely to buy items after seeing them in the IF, etc. However, implementing this method can be challenging and potentially risky. From a business perspective, deploying a potentially suboptimal recommendation system and allowing customers to interact with it may have undesirable consequences. Additionally, online testing is more complex than offline evaluation, requiring more time and resources. This is why the evaluation procedure in this work is only an approximation of how well a model performs with regard to different objectives but it can not replace the insights gained from conducting an online A/B-Test. This needs to be kept in mind as a model that appears superior based on offline evaluation could perform differently when interacting with real-time customers. This becomes especially relevant when considering that even though two models might show only slight differences in reported metrics, the actual recommendations presented to the customer could vary significantly, as only the top-12 predictions are showcased within IKEA's IF context.

Overall, the comprehensive analysis of the experiments investigating different window sizes revealed that the metrics proposed by Stamenkovic et al. [29], which are used throughout this work, are insufficient for the comparison of different models. In the given use case, they seemingly fail to catch obvious differences, especially concerning the diversity of model recommendations as the qualitative analysis in Chapter 6.3.2 clearly showcased.

7.3.1 Accuracy and Target Bias

For accuracy, HR and NDCG are used to measure how well the model is able to predict the next IF image click given the history of prior interactions by the customer. Although this approach is logical and provides valuable insights when applied to test datasets, the target itself includes a serious bias. The samples included in the dataset represent real customer interactions with the IKEA website. The supervised target used for prediction are the clicks on IF images, that are assumed to reflect customer preferences. But do these interactions actually represent the user's real preferences? As mentioned before, the IF shows twelve different images to the user from which they can choose one. The recorded data represents this choice - the preference of the customer among the twelve images shown to them given their previous history of interactions. However, it is entirely possible that given the choice from the entire selection of over 10 thousand images, the user might have chosen differently. Thus, the training target in this study is heavily biased towards the RS that was in place at the time the interactions were recorded.

As such, the computed accuracy metrics should be interpreted cautiously. Whether a trained model truly outperforms the model deployed during the data collection phase - in other words, whether it has learned to provide better recommendations to users - can only be validated through an online evaluation using actual customer engagement scores. Even if a model performs worse in the offline evaluation, it could be better with real interactions given the biased target.

In general, training a RS in this offline manner with the discussed targets is an iterative process involving deployment, interaction recording, offline training, A/B testing, and repeat. Given the outlined limitations, it is unrealistic to expect to train a flawless RS model in one go. The accuracy metrics calculated based on the biased target provide only a rough approximation of how effectively the RS captures customer preferences. It is essential for future research to explore evaluation methods that offer a more precise and nuanced understanding of the RS's with regard to this.

The biased target also makes it a lot harder for the model to actually pick up any long term relationships in the data, even if present. Consider the scenario where the prior RS that was used to record the datasets had a strong bias towards the latest few interactions of the customer. For instance, suppose a user first browsed beds and then proceeded to look at mirrors. A model heavily influenced by the latest interaction would suggest a range of IF images featuring mirrors in various room setups like living rooms, bathrooms, and so forth. Consequently, the twelve recommendations presented to the user in this context would predominantly include these general mirror images. But given the prior behavior of the customer in the full last ten timesteps, it might be more suitable to recommend images of bedrooms also containing mirrors. This would combine the two interests present in the prior interactions and might represent even better recommendations than just images focusing on mirrors in general. To accomplish this, the model needs to be able to learn this long-range dependency between the first and last interactions. However, since the customer was only presented with a selection of twelve images generated by the old model, which solely focused on the last mirror-views, they were limited to choosing their favorite picture from this restricted set of options. When using this interaction as a target for training a new model, it would therefore suggest that again the last clicks by the user - the mirror views - contain enough information to accurately predict the next image chosen by the customer. Consequently, there would be no benefit for the model to incorporate the bedroom view into its consideration for the next

image recommendations, as they likely add no additional information to predict the "general" mirror image in the next time step. However, the real preferences in this example, including long-range dependencies, might have been a combination of beds and mirrors together. This again highlights the need for an iterative process that allows models to progressively improve while allowing more long-term dependencies to trickle down into the (biased) target. This again allows the training of a potentially better RS, and so on.

One factor that aggravates the problem of biased targets is the fact that a further analysis of the data structure revealed that customer clicks on IF images are not only registered on the main IF, where twelve different recommendations are shown to the customer, but also in the context of the Product Information Page (PIP). The PIP is the page that users land on, when viewing a certain item and it only displays four IF image recommendations. This limited range of choices amplifies target bias, as it restricts the user's selections. From these recorded clicks, it is even less possible to conclude the actual preferences of the customer, given the constrained choice between only four images instead of the broader selection of twelve on the main IF feed. For further experiments, these PIP interactions should be removed entirely, reducing the already existing target bias in the data.

Another key point relevant to the RL component in the models is the absence of negative learning signals, as discussed in Chapter 5.1.2. The dataset only contains positive interactions, i.e., clicks on IF images, and does not record instances where the customer did not click on an IF despite it being displayed. Addressing this gap could provide a promising direction for improvement in the RL process.

7.3.2 Diversity and Novelty

For diversity and novelty, the two correlated coverage metrics proposed by Stamenkovic et al. [29] are used. As investigated in previous chapters, it is questionable if this fairly simple metric design really captures the true behavior of the model concerning these two objectives. Since literature suggests that both are important characteristics impacting customer satisfaction [5], [4], future research should explore alternative methods for measuring these traits, particularly considering the offline nature of the problem.

As these metrics were used to compare model classes, it is worth considering that alternative metrics could potentially impact these results, as it might reveal that recommendations by SMORL were, in fact, more diverse or novel compared to GRU4Rec.

A starting point for this could be to calculate the average cosine-similarity of IF images included in the top-12 predictions based on the learned vector representations discussed in 6.1. This would approximate how similar on average the different top recommendations are, with a lower score indicating a more diverse prediction batch.

7.3.3 Repetitiveness

The repetitiveness metric introduced in this work is also just an approximation of the real characteristics of the underlying RS. However, it only captures one of the two identified types of repetition in the IF. As revealed in the embedding analysis in Chapter 6.1, this metric fails to reflect the actual perceived repetitiveness. For instance, some IF images display the same room setup from different angles or zoom levels, yet these are labelled differently and thus treated as distinct images by the metric.

Further research should be done to adapt the metric in a way that it approximates the real underlying repetition behavior of the model, even in an offline setting.

8 Ethical considerations

"Personal data shall be processed lawfully, fairly and in a transparent manner in relation to the data subject."

- Article 5 (1a), EU General Data Protection Regulation[1]

As the use of RS by companies that shape the experience of customers with their various different services and platforms has become more and more popular, it also gets increasingly important to address the ethical considerations connected to these technologies. In this chapter, some of the ethical concerns associated with the development and deployment of RS are discussed, particularly given the use case underlying this work.

One of the most crucial moral considerations that needs to be taken into account is referring to the data used for the training of the RS. As shown in this work, these systems work by gathering user data to tailor their experiences to their personal preferences. Especially when using deep learning based approaches, it is particularly important to note that these methods require a substantial amount of data for training to achieve accurate and reliable results. In general, this data might contain personal details, browsing habits, buying history, and all sorts of other interactions with the platform. For the given use case, the data represents users clickstream on the IKEA website with special focus on the IF image interactions. Naturally, collecting this kind of information raises privacy concerns, since it could lead to misuse or unauthorized access to sensitive data. This is especially true when not only the raw interactions with the website are considered but if they can, at the same time, be connected to a users profile on the website for example, which could be used to clearly identify the person behind those clicks. That is why it is crucial for businesses using these systems to adopt transparent data collection practices and solid security measures to ensure user privacy.

The data for this research only included masked user IDs and was focused on personalizing the interactions of the customer with the IF solely based on previous clicks, without utilizing or having access to any further personal information that could be used to identify a user. Even though including this type of data in the modeling process could potentially lead to even more tailored recommendations and thus higher customer satisfaction, their use

[1] https://gdpr-info.eu/art-5-gdpr/

must be carefully evaluated due to the ethical considerations and imminent privacy risks mentioned above.

In general, transparency in this context is vital for companies. This ethical principle is also reflected by the citation of article 5 (1a) of the European Regulation at the start of this chapter. It must be clear to customers, when they decide to use a certain website or platform, which data they allow the company to process, store and use after their visit. This way, the trust of customers in a company can be increased, contributing to a positive customer relationship. In the long run, this might be an even more fruitful and valuable asset than developing ever more personalized recommendations given the discussed privacy risk and ethical concerns.

Another risk that must be considered in the context of RS are potential biases in the trained models. Especially when using deep learning approaches, which most of the time still represent *black boxes* whose decisions commonly lack explainability, biases can be introduced in such a system, either by accident as a bias might already be present in the training data, or on purpose by a person with harmful intentions. Even though this might not be of particular importance for the given use case, i.e., image recommendations for furnishing ideas, but for other services this could be different. For example RSs in the context of a social media platform like Twitter or Facebook could be used in a harmful way to introduce a bias that overrepresents posts from a specific political party or a fake news channel in their feed, potentially leading to an artificial influence on opinions or even radicalization.

As the discussion revealed, especially in the context of RS based on "data hungry" deep learning models, there exist a number of fundamental ethical challenges that must be considered before training and deploying such a model.

9 Conclusion

This work mainly focused on investigating the effectiveness of hybrid models combining classical Supervised Learning approaches with RL techniques applied to the use case of training a RS for IKEA's IF. For this purpose, the following three closely related models were explored:

1. **GRU4Rec:** A representative of purely Supervised Learning approaches, which was employed as a benchmark for the other RL-based models.
2. **SQN**: A model, which largely mirrors GRU4Rec's architecture, but incorporates a Q-learning-based RL component. This RL element serves a regularization function, aiming to enhance the model's performance.
3. **SMORL**: An extended version of SQN which incorporates two additional Q-heads. These are designed to promote the generation of more diverse and novel recommendations.

After a hyperparameter search of each model class based on the validation dataset, the most advantageous parameter configurations were determined for each class: GRU4Rec, SQN and SMORL. Utilizing these configurations, four models for each class were trained, resulting in a total of twelve trained models. These models were subsequently evaluated against the test dataset to compare the average performance of each model class in relation to the different objectives accuracy, novelty and diversity. Based on the analysis of these results, implications for the two first research questions were derived. These are briefly summarized in the following. Note, that the headlines only capture the core part of the questions, the full ones can be found in the introductory chapter.

RQ1: Does incorporating a RL component within a supervised model architecture improve next image accuracy (GRU4Rec vs. SQN)?

This research question essentially targets the comparison of GRU4Rec and SQN, as both share a similar architecture but the latter introduces RL into the learning procedure as a form of regularization [39]. As all other parameters and architectural components were kept the same, this allowed for an isolated investigation of the impact of RL on the model's training procedure. Contrary to the authors' claims that *"the proposed SQN method achieves consistently better performance than the corresponding baseline"* [39], the model performed worse in every assessed

dimension, including accuracy, diversity, novelty and repetitiveness. While these declines were minimal and the model still performed comparably to GRU4Rec, there is no clear evidence to support that incorporating RL into the GRU4Rec architecture brings any benefits for the particular use case at hand. The findings rather suggest that the RL component might disrupt the learning process, consequently lowering performance.

RQ2: How does the inclusion of two additional Q-heads for novelty and diversity impact recommendation performance across evaluated dimensions? (GRU4Rec/SQN vs. SMORL)?

The second research question builds on the first one and essentially targets the comparison of SQN with SMORL, as the SMORL model features two additional Q-heads in its RL part that were specifically designed to reinforce novel and diverse recommendations. However, given SQN's lower performance across all evaluation dimensions compared to GRU4Rec, the comparison of GRU4Rec and SMORL is considered for this question. Even though in their work the authors state that SMORL improved performance in every reported metric for accuracy, diversity, novelty and even repetitiveness [29], this was not the case for the conducted experiments in this work. Overall, SMORL performed on average comparable to GUR4Rec, while showing minor improvements for accuracy and at the same time decreased levels of diversity, novelty and repetitiveness. Even with a higher weighting on novelty and diversity in the hyperparameter search experiments, there was no evidence that this shifted the two objectives in an advantageous way, given the defined validation metrics adopted from the paper [29]. As we observed in the subsequent analysis of the conducted experiments, the stronger the influence of the RL in general, the worse was performance across all dimensions (except repetitiveness). While SMORL directly includes the two new objectives diversity and novelty into the optimization procedure, there is no evidence for the given experiments and the use case at hand that this actually lead to an improvement in these dimensions. In fact, experiment results rather suggest the opposite.

Given SMORL's added complexity in terms of data preparation, training procedure, and hyperparameter space, and its lack of significant performance improvement, there is insufficient justification for preferring it over GRU4Rec within the context of IKEA's IF.

RQ3: Does incorporating a larger window size improve model performance? And what is a sufficient window size for accurate predictions?

The results of the quantitative analysis of the experiments targeting different window sizes demonstrated a clear positive correlation between the size of the incorporated window and the accuracy of next image predictions. However, the improvements for larger window sizes were surprisingly moderate. This suggests that either the target used in the analysis primarily exhibits mild long-term relationships or that the data structure provided may not have enabled the model to effectively capture them.

The metrics connected to diversity and novelty of recommendation batches did not show any clear pattern and were similar for all experiments.

As the subsequent qualitative analysis of actual recommendations for two example cases revealed, in contrast to the changes represented in the metrics, recommendations differed significantly. Larger window sizes seemed to yield a much more diverse set of top-predictions, covering different potential areas of interest for the customer.

The results emphasize the need for more sophisticated metrics that are able to capture the actual recommendation behavior of different models, to allow for a more insightful offline comparison. Additionally, the results also underscore the potential necessity for adjustments in the data structure or model architecture, enabling models to effectively capture and leverage potential long-term dependencies in customer click-streams.

The model utilizing a window size of 5 exhibited notably diverse sets of recommendations for the two examples investigated. Therefore, considering the specific scenario and model structure, a window size of 5 is recommended as a promising starting point for future experiments.

Summary of Results

As the first two research questions revealed, the experiments showed no consistent evidence, that for the given use case and the GRU4Rec model as a baseline, adding RL to the architecture, either in form of SQN or SMORL, helps the model to learn better recommendations through the reinforcing reward signal. It remains to be explored whether, under what circumstances and for which objectives this approach could prove beneficial. As the conducted experiments suggest, for IKEA's IF and the objectives of diversity and novelty, the proposed procedure in the SMORL paper [29] did not provide an improvement in novelty and diversity.

This is why, based on these findings and supported by a generally lower complexity of the model, GRU4Rec is suggested as a model candidate for IKEA's IF, when considering the three different model classes investigated in this work.

For the future, exploring other models through further experiments could prove beneficial, and further research should be aimed at identifying refined metrics that can accurately reflect the actual behavior of the RS, even in the given offline setting.

As the qualitative analysis of the results revealed, a perhaps even more important focus than investigating new models appears to be a thorough consideration of changes to the data structure itself. The primary objective should be the reduction of target bias, the careful design of input sequences including the elimination of the redundant chain of products (e.g., for different colors), to increase degree of information and the mitigation of popularity bias potentially through undersampling of high frequency images. The overall goal is to provide learning systems with input data and targets that enable them to more effectively capture potential long-term dependencies in customer interactions to leverage their full potential, ultimately delivering a truly inspiring and engaging experience.

Bibliography

[1] Himan Abdollahpouri. "Popularity bias in ranking and recommendation". In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 2019, pp. 529–530.

[2] M Mehdi Afsar, Trafford Crump, and Behrouz Far. "Reinforcement learning based recommender systems: A survey". In: *ACM Computing Surveys* 55.7 (2022), pp. 1–38.

[3] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hachette UK, July 2006. ISBN: 978-1-4013-8463-0.

[4] Pablo Castells, Neil Hurley, and Saul Vargas. "Novelty and diversity in recommender systems". In: *Recommender systems handbook*. Springer, 2022, pp. 603–646.

[5] Pablo Castells, Saúl Vargas, and Jun Wang. "Novelty and diversity metrics for recommender systems: choice, discovery and relevance". In: (2011).

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*. Sept. 2014. DOI: 10.48550/arXiv.1406.1078. arXiv: 1406.1078 [cs, stat]. (Visited on 04/26/2023).

[7] Paul Covington, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations". In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[8] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Hado Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010. (Visited on 05/05/2023).

[11] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. "Session-based recommendations with recurrent neural networks". In: *arXiv preprint arXiv:1511.06939* (2015).

[12] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[14] Rong Hu and Pearl Pu. "Helping users perceive recommendation diversity." In: *DiveRS@ RecSys*. 2011, pp. 43–50.

[15] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[16] Wang-Cheng Kang and Julian McAuley. "Self-attentive sequential recommendation". In: *2018 IEEE international conference on data mining (ICDM)*. IEEE. 2018, pp. 197–206.

[17] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[18] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. "Conservative q-learning for offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[19] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[20] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. "Monolith: Real Time Recommendation System With Collisionless Embedding Table". In: *arXiv preprint arXiv:2209.07663* (2022).

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. Dec. 2013. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602 [cs]. (Visited on 05/05/2023).

[22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. *Training Language Models to Follow Instructions with Human Feedback*. Mar. 2022. arXiv: 2203.02155 [cs]. (Visited on 05/02/2023).

[23] Aske Plaat. *Deep Reinforcement Learning*. Springer, 2022.

[24] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Recommender systems: introduction and challenges". In: *Recommender systems handbook* (2015), pp. 1–34.

[25] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. "Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems". In: *Fifth International Conference on Computer and Information Science* (Jan. 2002).

[26] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. "Collaborative filtering recommender systems". In: *The adaptive web: methods and strategies of web personalization* (2007), pp. 291–324.

[27] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. (Visited on 05/02/2023).

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[29] Dusan Stamenkovic, Alexandros Karatzoglou, Ioannis Arapakis, Xin Xin, and Kleomenis Katevas. "Choosing the Best of Both Worlds: Diverse and Novel Recommendations through Multi-Objective Reinforcement Learning". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 2022, pp. 957–965.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction*. MIT Press, Nov. 2018. ISBN: 978-0-262-35270-3.

[31] Jiaxi Tang and Ke Wang. "Personalized top-n sequential recommendation via convolutional sequence embedding". In: *Proceedings of the eleventh ACM international conference on web search and data mining*. 2018, pp. 565–573.

[32] Peter D Turney and Patrick Pantel. "From frequency to meaning: Vector space models of semantics". In: *Journal of artificial intelligence research* 37 (2010), pp. 141–188.

[33] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. Dec. 2015. DOI: 10.48550/arXiv.1509.06461. arXiv: 1509.06461 [cs]. (Visited on 05/05/2023).

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[35] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. "A survey on session-based recommender systems". In: *ACM Computing Surveys (CSUR)* 54.7 (2021), pp. 1–38.

[36] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. *Dueling Network Architectures for Deep Reinforcement Learning*. Apr. 2016. arXiv: 1511.06581 [cs]. (Visited on 05/05/2023).

[37] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[38] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. "A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[39] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. "Self-supervised reinforcement learning for recommender systems". In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 931–940.

[40] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. "Supervised Advantage Actor-Critic for Recommender Systems". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 2022, pp. 1186–1196.

[41] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. "A simple convolutional generative network for next item recommendation". In: *Proceedings of the twelfth ACM international conference on web search and data mining*. 2019, pp. 582–590.

[42] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. "Deep learning based recommender system: A survey and new perspectives". In: *ACM computing surveys (CSUR)* 52.1 (2019), pp. 1–38.

A Appendix

A.1 Equations

A.1.1 Derivation of MSE loss from log-likelihood

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \quad \log \mathcal{L}(\theta; Y, X, \sigma) \tag{A.1}$$

$$= \underset{\theta}{\text{argmax}} \quad \log \left[\frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \right) \right] \tag{A.2}$$

$$= \underset{\theta}{\text{argmax}} \quad \log \left(\frac{1}{(\sigma\sqrt{2\pi})^n} \right) + \left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \right) \tag{A.3}$$

$$= \underset{\theta}{\text{argmax}} \quad -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \tag{A.4}$$

$$= \underset{\theta}{\text{argmin}} \quad \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \tag{A.5}$$

$$= \underset{\theta}{\text{argmin}} \quad \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \tag{A.6}$$

A.1.2 Derivation of Bellman Equation

In the following the Bellman Equation given in 2.42 is derived. We will start by splitting the expected value in equation 2.40 in two terms, which we will solve separately to then combine them in the end. As described in Equation 2.37, we will also use a short form of the full expected value

$$\mathbb{E}_{a\sim\pi(a|s),\ s',r\sim p(s',r|s,a)} \big[G_t | S_t = s \big], \tag{A.7}$$

but compared to the book [30] we will keep p indicating the MDP dynamics as an index for the expected value to make it clearer, that $s', r \sim p\,(s', r \mid s, a)$. So the following will be used as a short version

$$\mathbb{E}_{\pi,p} \big[G_t | S_t = s \big]. \tag{A.8}$$

$$v_\pi(s) = \mathbb{E}_{\pi,p}\left[G_t \mid S_t = s\right] \tag{A.9}$$
$$= \mathbb{E}_{\pi,p}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \tag{A.10}$$
$$= \underbrace{\mathbb{E}_{\pi,p}\left[R_{t+1} \mid S_t = s\right]}_{A} + \underbrace{\mathbb{E}_{\pi,p}\left[\gamma G_{t+1} \mid S_t = s\right]}_{B} \tag{A.11}$$

We will start with term A. For the summing indices we will also use a shorter form assuming that $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}$.

$$\mathbb{E}_{\pi,p}\left[R_{t+1} \mid S_t = s\right] = \sum_r r p(r|s) \tag{A.12}$$
$$= \sum_r \sum_{s'} \sum_a r p(r, s', a|s) \tag{A.13}$$
$$= \sum_r \sum_{s'} \sum_a r p(s', r|s, a)\pi(a|s) \tag{A.14}$$

Note that to get to equation A.13 we use the fact that the probability distribution $p(r|s)$ over the rewards r given state s is a marginal distribution of the joint probability $p(r, s', a|s)$ of r, s', a conditioned on the current state s. This joint distribution can then be further split into the different conditional distributions in A.14 using the chain rule of probability and the previously defined distributions p and π.

Now we proceed with the second term B. We follow the same logic as for A, by expressing the distribution we are looking for as the marginal of the known conditionals of the joint probability.

$$\mathbb{E}_{\pi,p}\left[\gamma G_{t+1} \mid S_t = s\right] = \gamma \sum_{g_{t+1}} g_{t+1} p(g_{t+1}|s) \tag{A.15}$$
$$= \gamma \sum_{g_{t+1}} \sum_r \sum_{s'} \sum_a g_{t+1} p(g_{t+1}, s', r, a|s) \tag{A.16}$$
$$= \gamma \sum_{g_{t+1}} \sum_r \sum_{s'} \sum_a g_{t+1} p(g_{t+1}|s', r, a, s) p(r, s', a|s) \tag{A.17}$$
$$= \gamma \sum_{g_{t+1}} \sum_r \sum_{s'} \sum_a g_{t+1} p(g_{t+1}|s', r, a, s) p(s', r|s, a)\pi(a|s) \tag{A.18}$$
$$\overset{\text{MP}}{=} \gamma \sum_{g_{t+1}} \sum_r \sum_{s'} \sum_a g_{t+1} p(g_{t+1}|s') p(s', r|s, a)\pi(a|s) \tag{A.19}$$
$$\overset{\text{Eq. A.15}}{=} \gamma \sum_r \sum_{s'} \sum_a \mathbb{E}_{\pi,p}\left[\gamma G_{t+1} \mid S_{t+1} = s'\right] p(s', r|s, a)\pi(a|s) \tag{A.20}$$
$$= \gamma \sum_r \sum_{s'} \sum_a v_\pi(s') p(s', r|s, a)\pi(a|s) \tag{A.21}$$

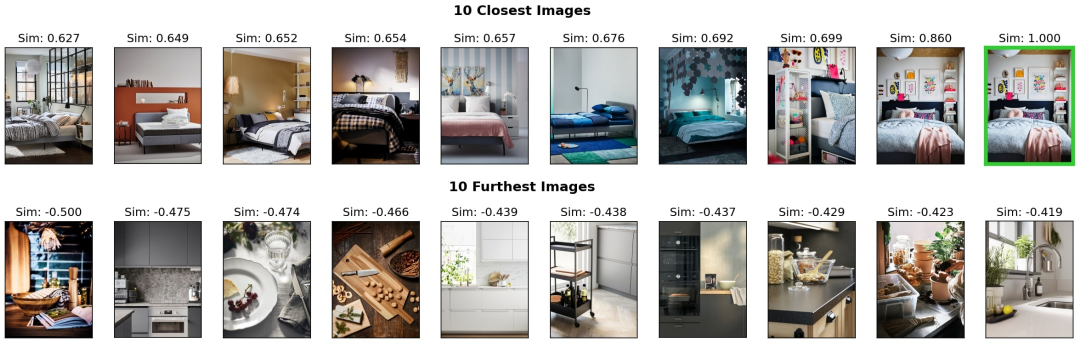
For the step in A.19, to get from $p(g_{t+1}|s', r, a, s)$ to $p(g_{t+1}|s')$, we use the Markov property introduced in Chapter 2.4.1. The conditional probability $p(g_{t+1}|s', r, a, s)$ describes the distribution of the cumulated future reward g_{t+1} from timestep $t+1$ onward given s, a, r, s'. From the Markov property we know, that given the next state s', the previous state s, action a and reward r, do not influence the reward for the next timestep $t+1$ and the steps after that until termination. So given s', $p(g_{t+1}|s', r, a, s)$ is independent from s, a, r and can thus be simplified to $p(g_{t+1}|s')$. After that, to get to the expression in A.20, we just need to see, that part of the expression is similar to A.15 and can be replaced by the expectation. Now we just need to use the definition of the value function introduced in equation 2.39 and insert it.

Finally, the two parts from A and B can be put together, to form the famous Bellman equation:

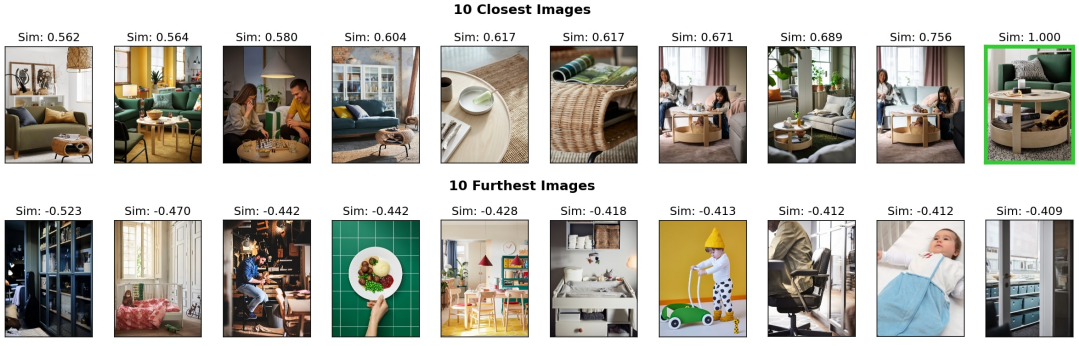
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_{\pi,p}\left[G_t \mid S_t = s\right] && \text{(A.22)}\\
&= \mathbb{E}_{\pi,p}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \text{(A.23)}\\
&\overset{\text{A,B}}{=} \sum_r \sum_{s'} \sum_a \left[r + \gamma v_\pi(s')\right] p(s', r|s, a)\pi(a|s) && \text{(A.24)}
\end{aligned}
$$

A.2 Result Visualizations and Tables

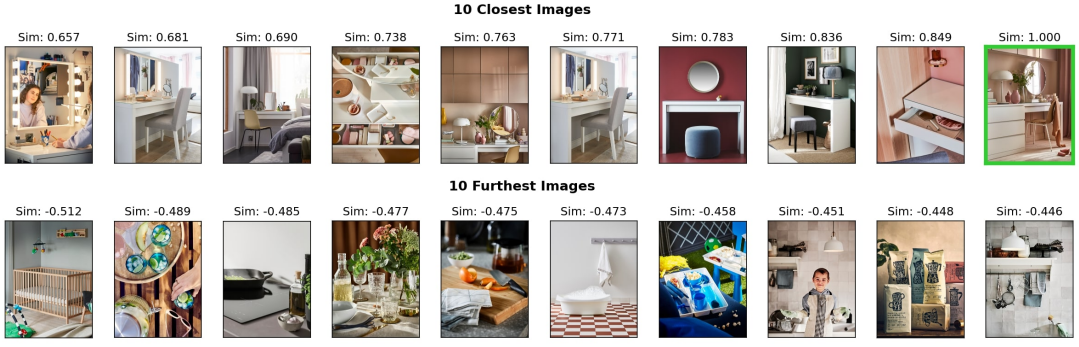
(a)



(b)



(c)

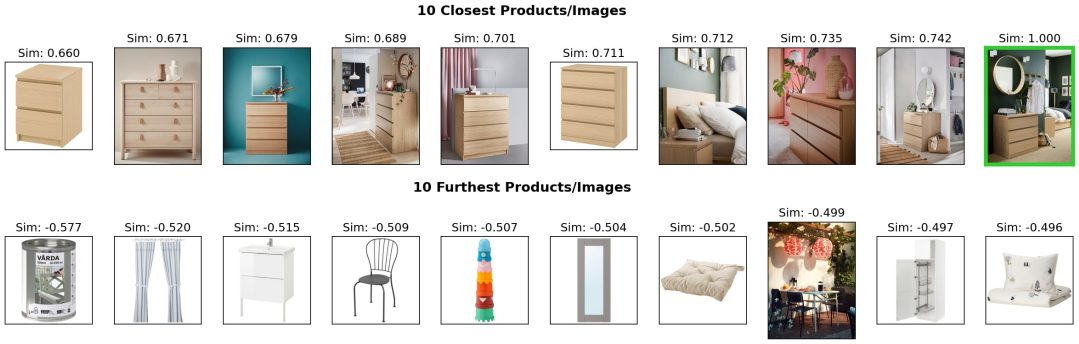


(d)

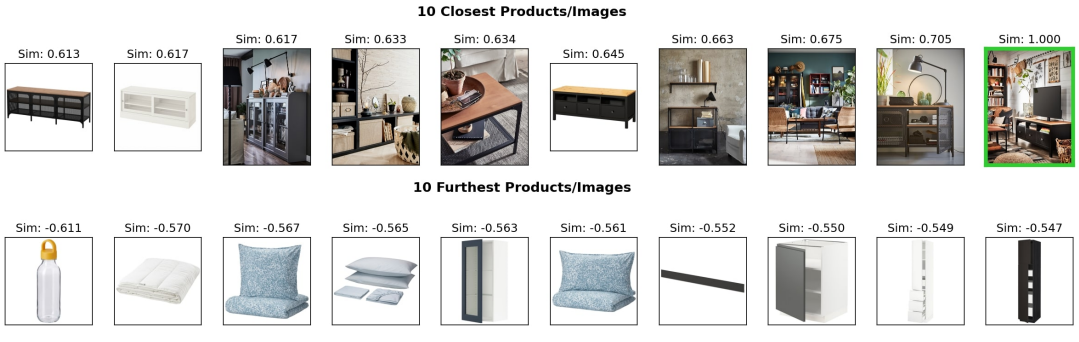
Figure A.1: Further example embedding visualization of pretrained embedding layer - only including IF images

(a)

(b)



(c)



(d)

Figure A.2: Further example embedding visualization of pretrained embedding layer - IF images and products

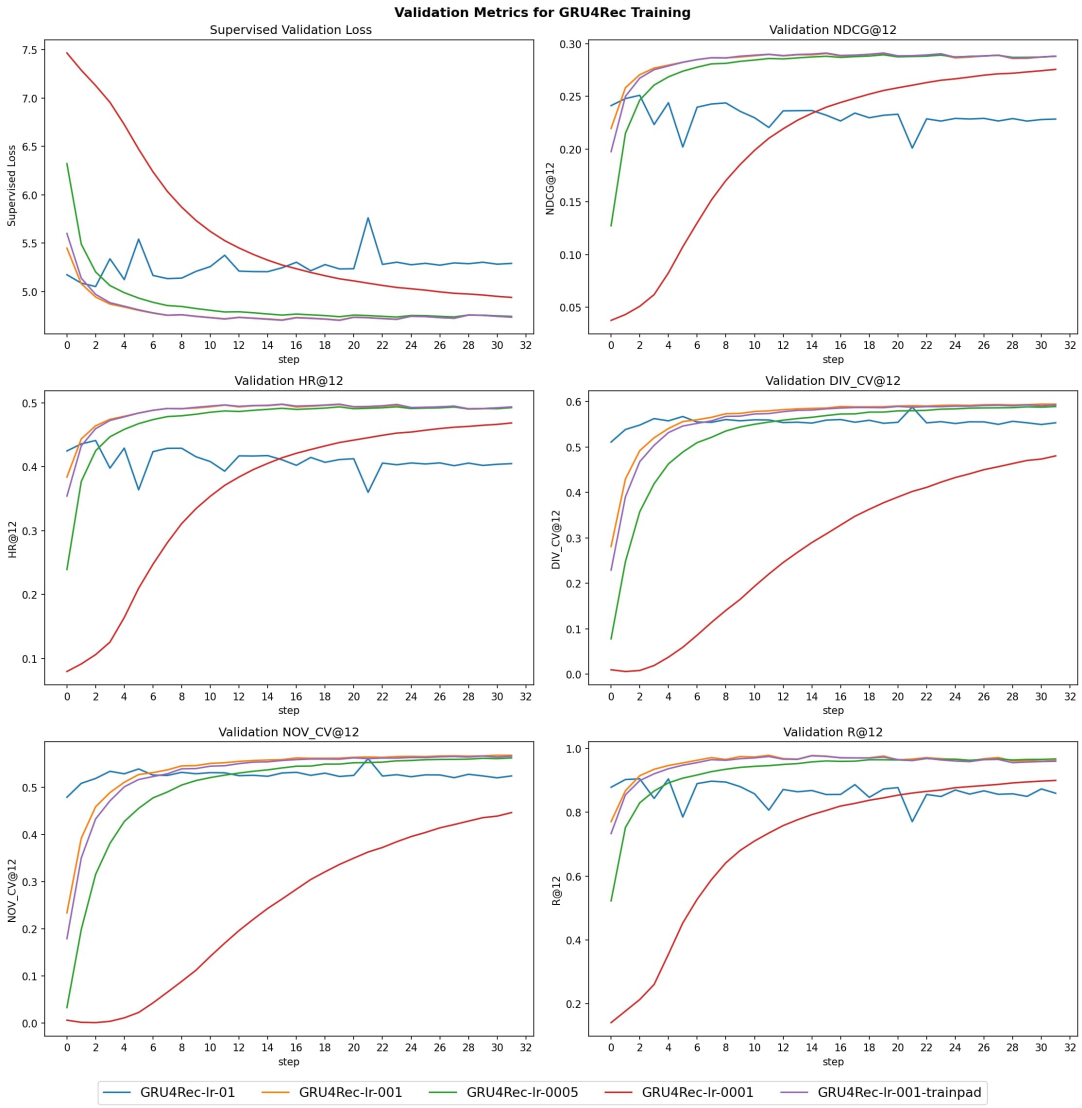


Figure A.3: Validation metrics for GRU4Rec hyperparameter search runs

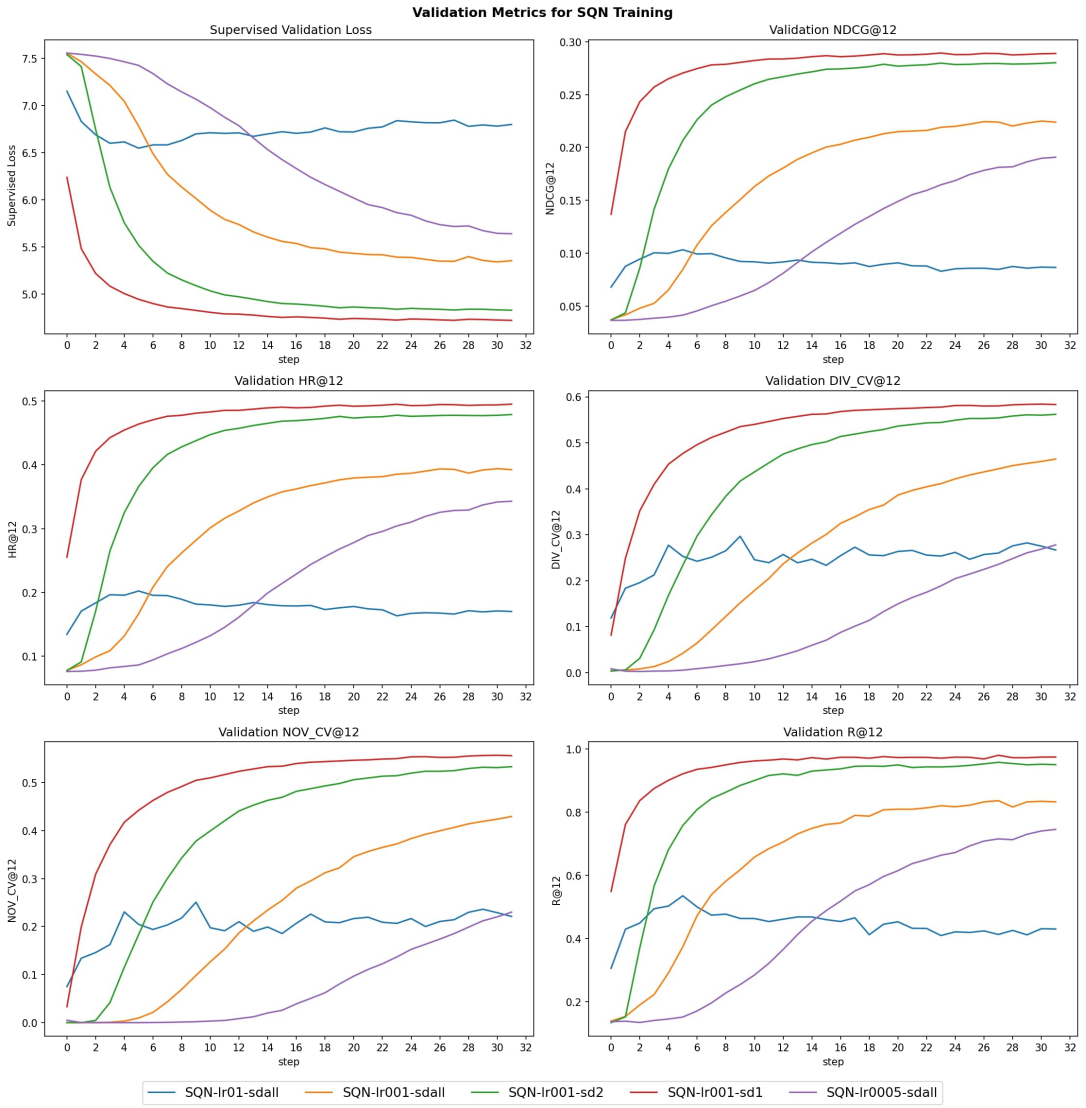


Figure A.4: Validation metrics for SQN hyperparameter search runs

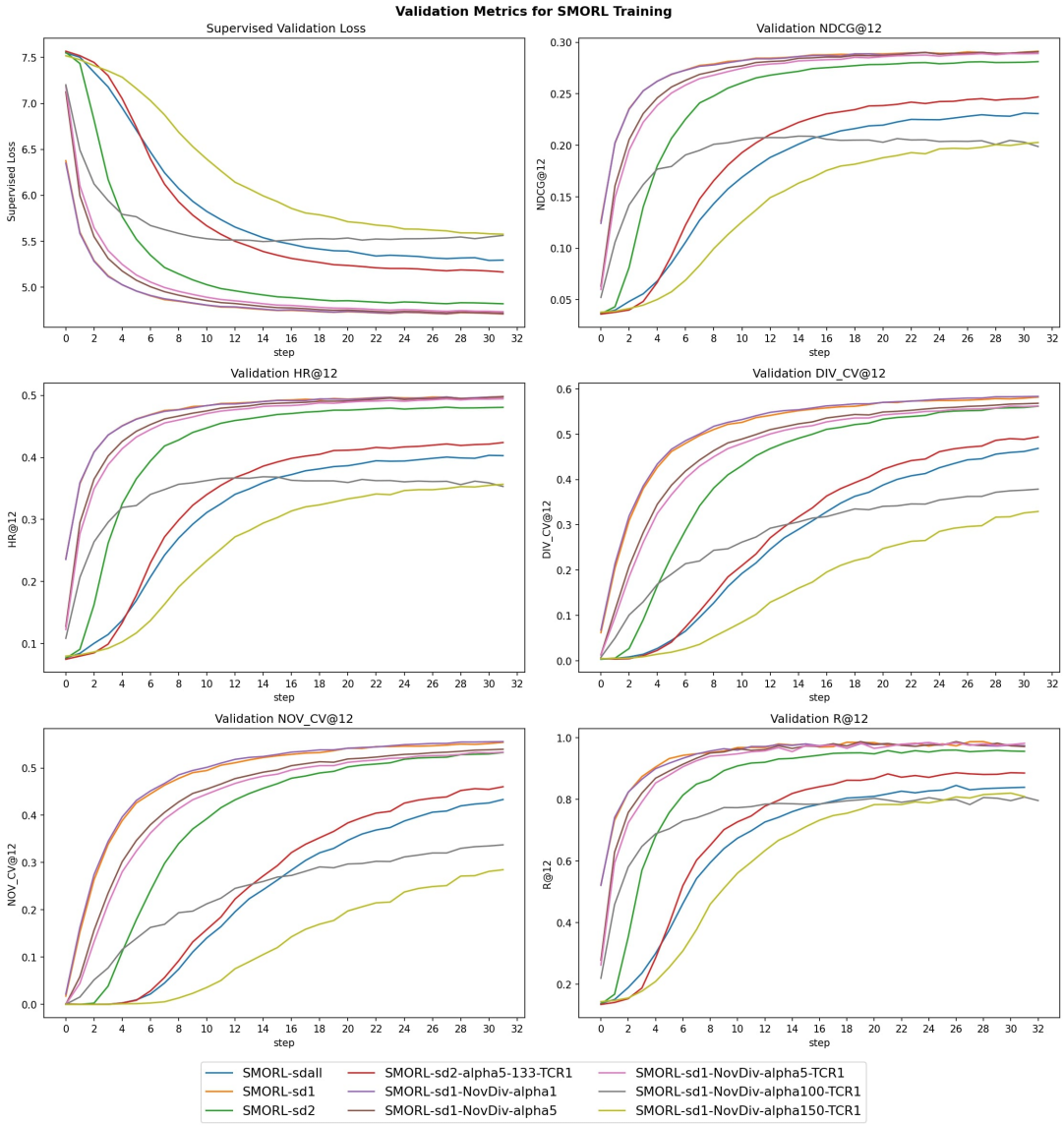


Figure A.5: Validation metrics for SMORL hyperparameter search runs

Table A.1: Train metrics for GRU4Rec hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
GRU4Rec-lr-01	5.0663	0.1250	0.3550	0.4696	0.1250	0.2390	0.2735	0.5497	0.5784	0.5862	0.5198	0.5505	0.5588	0.2689	0.7342	0.9154
GRU4Rec-lr-001*	4.5760	0.1487	0.4061	0.5260	0.1487	0.2768	0.3130	0.5532	0.5875	0.5942	0.5235	0.5602	0.5674	0.2911	0.8080	0.9829
GRU4Rec-lr-0005	4.6074	0.1481	0.4023	0.5207	0.1481	0.2747	0.3104	0.5239	0.5755	0.5825	0.4923	0.5474	0.5548	0.3042	0.8067	0.9719
GRU4Rec-lr-0001	4.8637	0.1357	0.3717	0.4842	0.1357	0.2532	0.2871	0.3599	0.4585	0.4848	0.3179	0.4225	0.4506	0.3340	0.7606	0.9001
GRU4Rec-lr-001-trainpad	4.5314	0.1528	0.4125	0.5325	0.1528	0.2821	0.3184	0.5541	0.5892	0.5946	0.5245	0.5620	0.5678	0.2888	0.8058	0.9841

Table A.2: Validation metrics for GRU4Rec hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
GRU4Rec-lr-01	5.0533	0.1088	0.3274	0.4410	0.1088	0.2169	0.2511	0.4158	0.5262	0.5484	0.3773	0.4947	0.5185	0.2634	0.7249	0.9053
GRU4Rec-lr-001*	4.7082	0.1327	0.3789	0.4974	0.1327	0.2549	0.2907	0.4971	0.5768	0.5858	0.4637	0.5488	0.5584	0.2940	0.7972	0.9651
GRU4Rec-lr-0005	4.7415	0.1338	0.3764	0.4936	0.1338	0.2541	0.2895	0.4869	0.5659	0.5770	0.4528	0.5372	0.5490	0.3312	0.7590	0.9002
GRU4Rec-lr-0001	4.9402	0.1283	0.3574	0.4684	0.1283	0.2422	0.2757	0.3437	0.4536	0.4807	0.3008	0.4173	0.4462	0.2742	0.7897	0.9732
GRU4Rec-lr-001-trainpad	4.7040	0.1338	0.3791	0.4978	0.1338	0.2554	0.2912	0.5065	0.5773	0.5867	0.4738	0.5493	0.5593	0.2736	0.7955	0.9764

Table A.3: Train metrics for SQN hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SQN-lr01-sdall	6.5774	0.0360	0.1344	0.2028	0.0360	0.0834	0.1039	0.2185	0.3224	0.3689	0.1694	0.2777	0.3270	0.1040	0.3614	0.5150
SQN-lr001-sdall	5.2541	0.1037	0.3023	0.4059	0.1037	0.2018	0.2331	0.3846	0.4630	0.4872	0.3439	0.4274	0.4531	0.2592	0.6752	0.8381
SQN-lr0005-sdall	5.6563	0.0812	0.2527	0.3500	0.0812	0.1653	0.1947	0.1852	0.2633	0.2984	0.1337	0.2145	0.2517	0.2271	0.5889	0.7444
SQN-lr001-sd2	4.6968	0.1394	0.3852	0.5019	0.1394	0.2616	0.2968	0.5076	0.5605	0.5702	0.4749	0.5314	0.5417	0.3150	0.7942	0.9573
SQN-lr001-sd1*	4.5169	0.1526	0.4119	0.5318	0.1526	0.2817	0.3179	0.5421	0.5841	0.5906	0.5117	0.5566	0.5635	0.2842	0.8046	0.9834

Table A.4: Validation metrics for SQN hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SQN-lr01-sdall	6.5480	0.0352	0.1336	0.2021	0.0352	0.0827	0.1033	0.1203	0.2088	0.2530	0.0783	0.1597	0.2045	0.1125	0.3760	0.5357
SQN-lr001-sdall	5.2932	0.1021	0.2998	0.4024	0.1021	0.1999	0.2309	0.3470	0.4489	0.4774	0.3044	0.4123	0.4427	0.2523	0.6670	0.8331
SQN-lr0005-sdall	5.6405	0.0796	0.2474	0.3429	0.0796	0.1620	0.1908	0.1568	0.2412	0.2779	0.1068	0.1917	0.2300	0.2687	0.7907	0.9749
SQN-lr001-sd2	4.8227	0.1293	0.3660	0.4807	0.1293	0.2468	0.2814	0.4682	0.5502	0.5651	0.4329	0.5204	0.5362	0.2255	0.5890	0.7459
SQN-lr001-sd1*	4.7189	0.1324	0.3772	0.4960	0.1324	0.2540	0.2899	0.5010	0.5739	0.5836	0.4679	0.5456	0.5560	0.3070	0.7859	0.9509

Note: * indicates hyperparameter settings chosen for testing

Table A.5: Train metrics for SMORL hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SMORL-sdall	5.2354	0.1064	0.3086	0.4129	0.1064	0.2065	0.2379	0.3845	0.4621	0.4870	0.3438	0.4263	0.4529	0.2596	0.6824	0.8464
SMORL-sd2	4.6857	0.1400	0.3867	0.5042	0.1400	0.2627	0.2982	0.5028	0.5594	0.5690	0.4698	0.5302	0.5404	0.3190	0.7987	0.9619
SMORL-sd1*	4.5100	0.1525	0.4126	0.5334	0.1525	0.2819	0.3184	0.5335	0.5801	0.5867	0.5025	0.5523	0.5593	0.2882	0.8112	0.9886
SMORL-sd1-NovDiv-alpha1	4.5389	0.1497	0.4084	0.5287	0.1497	0.2784	0.3148	0.5303	0.5773	0.5869	0.4992	0.5493	0.5595	0.2863	0.8103	0.9862
SMORL-sd1-NovDiv-alpha5	4.5472	0.1486	0.4077	0.5280	0.1486	0.2775	0.3138	0.5025	0.5628	0.5745	0.4695	0.5338	0.5462	0.2999	0.8148	0.9858
SMORL-sd1-NovDiv-alpha5-TCR1	4.5737	0.1468	0.4038	0.5236	0.1468	0.2746	0.3108	0.4946	0.5575	0.5690	0.4610	0.5282	0.5404	0.3105	0.8180	0.9863
SMORL-sd1-NovDiv-alpha100-TCR1	5.4476	0.0930	0.2740	0.3740	0.0930	0.1821	0.2122	0.2083	0.2922	0.3272	0.1577	0.2452	0.2824	0.2599	0.6331	0.7856
SMORL-sd1-NovDiv-alpha150-TCR1	5.5591	0.0902	0.2649	0.3613	0.0902	0.1762	0.2052	0.2196	0.3059	0.3386	0.1692	0.2598	0.2946	0.2627	0.6553	0.8143
SMORL-sd2-alpha5-133-TCR1	5.1125	0.1164	0.3260	0.4322	0.1164	0.2204	0.2524	0.4078	0.4864	0.5039	0.3686	0.4523	0.4710	0.2946	0.7290	0.8902

Table A.6: Validation metrics for SMORL hyperparameter search

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SMORL-sdall	5.2896	0.1025	0.2999	0.4034	0.1025	0.2001	0.2313	0.3285	0.4335	0.4619	0.2851	0.3959	0.4261	0.2573	0.6738	0.8376
SMORL-sd2	4.8088	0.1290	0.3670	0.4820	0.1290	0.2471	0.2818	0.4601	0.5450	0.5605	0.4244	0.5148	0.5314	0.2710	0.7912	0.9554
SMORL-sd1*	4.7053	0.1337	0.3793	0.4984	0.1337	0.2554	0.2914	0.4917	0.5723	0.5820	0.4580	0.5439	0.5543	0.3098	0.7905	0.9733
SMORL-sd1-NovDiv-alpha1	4.7074	0.1331	0.3787	0.4980	0.1331	0.2549	0.2909	0.4912	0.5706	0.5810	0.4576	0.5421	0.5532	0.2723	0.7957	0.9764
SMORL-sd1-NovDiv-alpha5	4.7148	0.1333	0.3793	0.4974	0.1333	0.2553	0.2909	0.4662	0.5546	0.5683	0.4307	0.5250	0.5397	0.2869	0.7990	0.9709
SMORL-sd1-NovDiv-alpha5-TCR1	4.7268	0.1334	0.3777	0.4954	0.1334	0.2545	0.2900	0.4668	0.5525	0.5631	0.4314	0.5228	0.5341	0.2982	0.8103	0.9815
SMORL-sd1-NovDiv-alpha100-TCR1	5.4513	0.0950	0.2774	0.3772	0.0950	0.1850	0.2151	0.1833	0.2792	0.3161	0.1318	0.2318	0.2708	0.2596	0.6321	0.7837
SMORL-sd1-NovDiv-alpha150-TCR1	5.5737	0.0893	0.2618	0.3567	0.0893	0.1742	0.2028	0.2046	0.2955	0.3294	0.1543	0.2488	0.2848	0.2616	0.6512	0.8089
SMORL-sd2-alpha5-133-TCR1	5.1637	0.1135	0.3192	0.4240	0.1135	0.2154	0.2470	0.3808	0.4731	0.4940	0.3402	0.4381	0.4604	0.2907	0.7241	0.8850

Note: * indicates hyperparameter settings chosen for testing

Table A.7: Test metrics for GRU4Rec test runs

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
GRU4Rec-Test1	4.7168	0.1324	0.3772	0.4966	0.1324	0.2540	0.2900	0.5006	0.5747	0.5848	0.4675	0.5465	0.5573	0.2791	0.7930	0.9727
GRU4Rec-Test2	4.7241	0.1324	0.3776	0.4960	0.1324	0.2540	0.2898	0.4851	0.5676	0.5802	0.4509	0.5390	0.5524	0.2888	0.7976	0.9701
GRU4Rec-Test3	4.7193	0.1322	0.3768	0.4955	0.1322	0.2536	0.2895	0.5095	0.5796	0.5887	0.4770	0.5517	0.5614	0.2686	0.7901	0.9734
GRU4Rec-Test4	4.7149	0.1319	0.3775	0.4968	0.1319	0.2537	0.2897	0.4990	0.5737	0.5833	0.4658	0.5454	0.5556	0.2736	0.7907	0.9712
μ	**4.7188**	**0.1322**	**0.3773**	**0.4962**	**0.1322**	**0.2538**	**0.2897**	**0.4986**	**0.5739**	**0.5842**	**0.4653**	**0.5456**	**0.5567**	**0.2775**	**0.7928**	**0.9719**

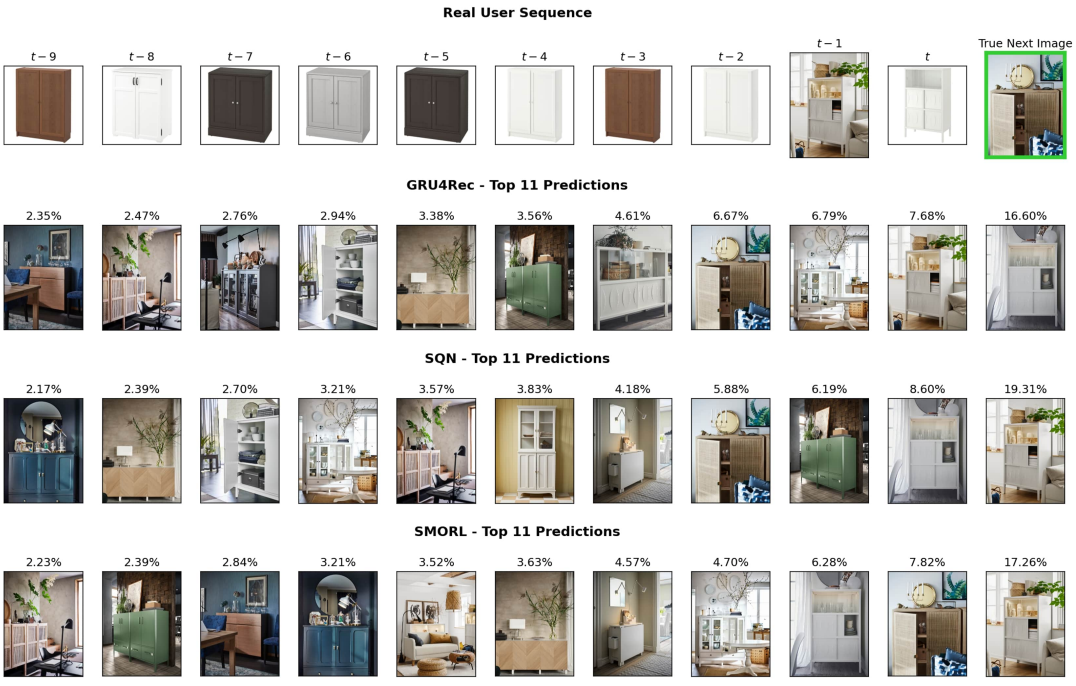
Table A.8: Test metrics for SQN test runs

		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SQN-Test1	4.7224	0.1323	0.3770	0.4957	0.1323	0.2537	0.2895	0.4953	0.5710	0.5834	0.4619	0.5425	0.5557	0.2707	0.7898	0.9747
SQN-Test2	4.7263	0.1319	0.3769	0.4956	0.1319	0.2534	0.2892	0.4944	0.5718	0.5824	0.4608	0.5434	0.5547	0.2720	0.7892	0.9725
SQN-Test3	4.7245	0.1319	0.3764	0.4955	0.1319	0.2531	0.2890	0.4965	0.5726	0.5831	0.4632	0.5442	0.5554	0.2697	0.7897	0.9760
SQN-Test4	4.7250	0.1324	0.3767	0.4952	0.1324	0.2536	0.2893	0.4937	0.5709	0.5823	0.4601	0.5424	0.5546	0.2766	0.7944	0.9758
μ	**4.7245**	**0.1321**	**0.3767**	**0.4955**	**0.1321**	**0.2534**	**0.2893**	**0.4950**	**0.5716**	**0.5828**	**0.4615**	**0.5432**	**0.5551**	**0.2722**	**0.7908**	**0.9748**

Table A.9: Test metrics for SMORL test runs

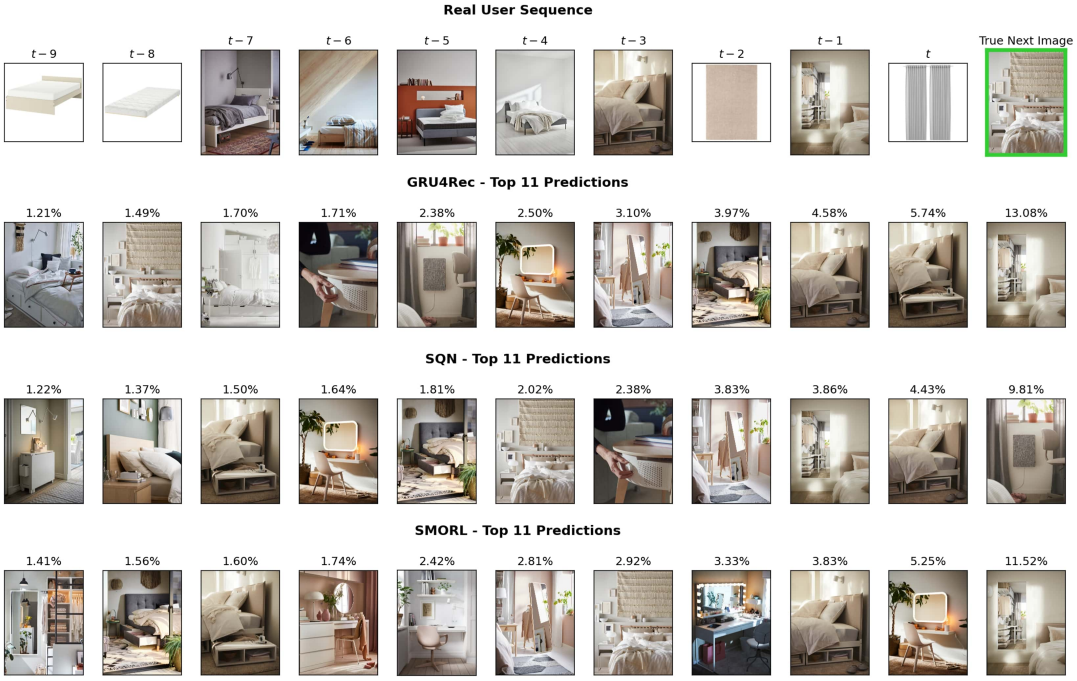
		HR			NDCG			CV_{div}			CV_{nov}			R		
Name	Loss	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12	@1	@6	@12
SMORL-Test1	4.7127	0.1326	0.3783	0.4972	0.1326	0.2546	0.2905	0.4833	0.5631	0.5750	0.4490	0.5341	0.5469	0.2820	0.7997	0.9781
SMORL-Test2	4.7108	0.1331	0.3786	0.4982	0.1331	0.2548	0.2909	0.4898	0.5703	0.5806	0.4559	0.5418	0.5528	0.2783	0.7953	0.9741
SMORL-Test3	4.7175	0.1331	0.3783	0.4963	0.1331	0.2548	0.2904	0.4821	0.5650	0.5766	0.4477	0.5361	0.5486	0.2765	0.7936	0.9729
SMORL-Test4	4.7144	0.1332	0.3789	0.4975	0.1332	0.2551	0.2909	0.4928	0.5690	0.5800	0.4591	0.5404	0.5522	0.2761	0.7871	0.9674
μ	**4.7138**	**0.1330**	**0.3785**	**0.4973**	**0.1330**	**0.2548**	**0.2907**	**0.4870**	**0.5668**	**0.5781**	**0.4529**	**0.5381**	**0.5501**	**0.2782**	**0.7939**	**0.9731**

(a) First Appendix example recommendations



(b) Second Appendix example recommendations

Figure A.6: Further recommendation comparison GRU4Rec, SQN and SMORL (1)



(a) Third Appendix example recommendations

(b) Fourth Appendix example recommendations

Figure A.7: Further recommendation comparison GRU4Rec, SQN and SMORL (2)

(a) Fourth Appendix example recommendations

(b) Fifth Appendix example recommendations

Figure A.8: Further recommendation comparison GRU4Rec, SQN and SMORL (3)

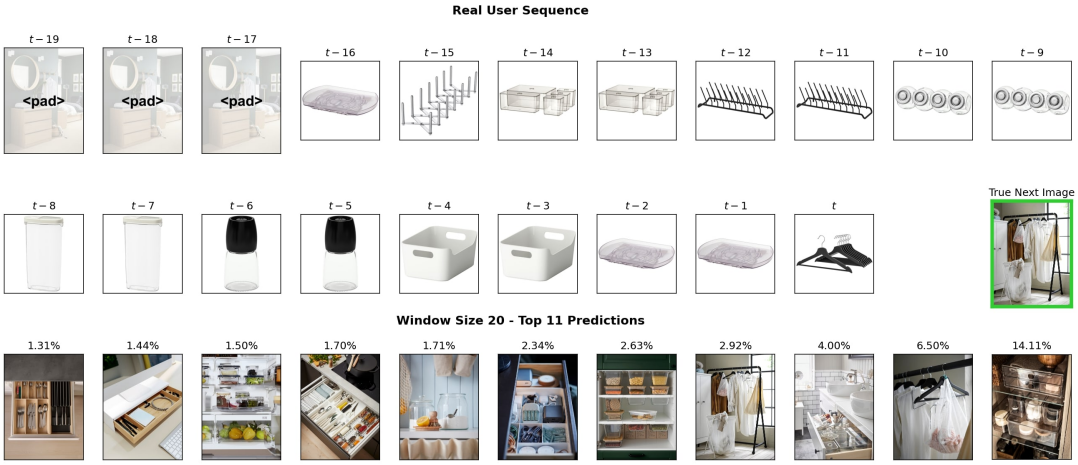


Figure A.9: GRU4Rec recommendations with window size 20

A.3 Code Snippets

```python
def calculate_session_repetitions(session_ids, top_20_preds):
    top_10_preds = [x[-10:] for x in top_20_preds]
    top_5_preds = [x[-5:] for x in top_20_preds]
    rpt_df = pd.DataFrame({
        'session_id': session_ids,
        'top_20_preds': top_20_preds,
        'top_10_preds': top_10_preds,
        'top_5_preds': top_5_preds
    })
    num_rpts_20 = 0
    num_rpts_10 = 0
    num_rpts_5 = 0
    rpt_groups = rpt_df.groupby('session_id')
    for _, group in rpt_groups:
        all_top_20 = np.concatenate(group.top_20_preds.values)
        all_top_10 = np.concatenate(group.top_10_preds.values)
        all_top_5 = np.concatenate(group.top_5_preds.values)
        num_rpts_20 += len(all_top_20) - len(np.unique(all_top_20))
        num_rpts_10 += len(all_top_10) - len(np.unique(all_top_10))
        num_rpts_5 += len(all_top_5) - len(np.unique(all_top_5))
    num_rpts_20 /= rpt_groups.ngroups
    num_rpts_10 /= rpt_groups.ngroups
    num_rpts_5 /= rpt_groups.ngroups
    return num_rpts_5, num_rpts_10, num_rpts_20
```

Figure A.10: Function from original SMORL paper [29] to compute average repetitions per session based on accumulated top-20 predictions for all sessions and samples in each session

```python
class Args:
    # ...
    r_click = 0.2
    r_buy = 1.0
    # ...
# ...

# Network parameters
args = Args()

# ...
reward_click = torch.repeat_interleave(torch.Tensor([args.r_click]), args.batch_size)
reward_buy = torch.repeat_interleave(torch.Tensor([args.r_buy]), args.batch_size)
# ...

for state, len_state, action, next_state, len_next_state, is_buy, is_done in train_loader:
    # ...
    acc_reward = torch.where(is_buy > 0, reward_buy, reward_click)

    supervised_out, smorl_loss = main_QN(
                state.to(device).long(),
                len_state.long(),
                action.to(device),
                acc_reward.to(device),
                target_Qs.to(device),
                target_Qs_selector.to(device),
                is_done.to(device)
            )
    # ...
```

Figure A.11: Code snippet from the original SMORL paper [29] that shows the computation of different rewards for clicks and purchases including how they get passed to the network.