AUTOSREC: AUTOMATED SEQUENTIAL RECOMMENDATION

A Thesis

by

SUIL LEE

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tie Liu
Co-Chair of Committee,	Xia Hu
Committee Members,	Serap Savari
	Raffaella Righetti
Head of Department,	Miroslav M Begovic

August 2021

Major Subject: Electrical Engineering

Copyright 2021 Suil Lee

ABSTRACT

Most traditional recommendation systems aim to discover users' preferences by predicting top scores on items. However, this recommendation system might not fully capture users' preferences because they do not involve any sequential information. Moreover, developing a sequential recommendation system is a complex task because the settings for each sequential scenario are very different. We present an automated sequential recommendation, AutoSRec, to unify such intricate tasks to address this problem.

Each sequential model shares some good attributes, and they have useful features to deal with their situations. However, a fine-grained neural model is not able to tackle all sequential recommendation problems. Our work aims to build upon three concepts: unification of sequential models, automation of recommendation systems, and user-friendly framework.

To achieve this, we extract critical components of recommendation models and combine them into what we call: hyper-interactions. Then, with the help of automated machine learning, AutoML, our AutoSRec can achieve the best model in the hyper-interaction search space. Lastly, AutoSRec is based on the TensorFlow API ecosystem, where even non-experts can understand quickly. Experiments on multiple data sets show the effectiveness of our approach.

ACKNOWLEDGMENTS

Many people helped me during my Master's study. I want to take this moment to thank them. I thank my advisor, Dr. Xia Hu, all my committee members, Dr. Tie Liu, Dr. Raffaella Righetti, and Dr. Serap Savari. I thank my lab mates and friends for their advice and support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Professor Jane Doe [advisor — also note if co-advisor] and John Doe of the Department of [Home Department] and Professor(s) XXXX of the Department of [Other Department].

The data analyzed for Chapter IV was provided by Professor Thompson. The analyses depicted in Chapter X were conducted in part by Daniel James of the Department of Statistics and were published in (2004) in an article listed in the Journal of Things.

All other work conducted for the thesis (or) dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from Texas A&M University and a dissertation research fellowship from That Foundation. OR No other outside source of funding was provided. One or the other must be stated.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Throughout the last decade, with the advance of e-commerce, recommendation system has more impact than before. A recommendation system is essential today. In general, the recommendation system's main task is to model interactions between users and items [1]. In research and industry fields, experts are constantly developing a better recommendation system. For example, Netflix offered a prize of 1 million dollars to whoever performs better compared to their algorithm. As follows, building a recommendation system and finding users' needs are very critical.

There are different approaches to build recommendation systems. Significantly, machine learning has been the most successful in recommendation systems. Machine learning is applied to find users' preferences and interests in products. Even though machine learning shows success in identifying users' needs, building such systems is problematic because it requires a solid understanding of machine learning and good programming skills. To alleviate building these recommendation systems, automated machine learning (AutoML) arose in the field.

1.1 Background

In this section, two topics are introduced: AutoML and recommendation system.

1.1.1 Automated Machine Learning (AutoML)

Machine learning has been emerging remarkably in the past few years. Tons of quality researches and discoveries are published every week. In the middle of this growth, a proliferation of automated machine learning, AutoML, is catching the attention of many experts. That is because AutoML significantly reduces the repetitive, time-consuming tasks of machine learning and improves the performance of training [2]. There are different aspects of AutoML, including model search, hyperparameters optimization, evaluation, training, and others. This research paper focuses on two aspects of AutoML: model search and hyperparameter optimization.

Model Search

The model search is an AutoML algorithm that examines the model structure. It helps to speed up the process of finding the right model for the task. Model search, specific for this research, chooses from a set of component blocks that consists of a deep neural network (DNN). Some examples of models are matrix factorization [3], self-attention [4], etc. The system evaluates a set of candidate models, then selects the best-performing model. The model search is an iterative step until the best model is found. Further, to reduce time searching for the best model, call-backs are used to skip unnecessary models while training.

Hyperparameter Optimization

Hyperparameter optimization is one of the essential tasks in AutoML. Hyperparameters are parameters that cannot be learned from the data and are external to the model. Every machine learning system has hyperparameters. There is a wide range of hyperparameters to optimize, such as learning rate, hidden layers, hidden units, number of epochs, etc. Fitted hyperparameters can significantly increase a model's performance. AutoML can automatically find these hyperparameters to improve the performance. The advantages of hyperparameter optimization in AutoML are (1) reduction of human and time effort to find good hyperparameters, (2) improvement of the performance, (3) improvement of reproducibility and fairness.

1.1.2 Recommendation System

Overview of Recommendation System

A recommendation system, often called a recommenders system, is a popular research topic. It recommends valuable items to users by understanding their behaviors. The steps of recommendation systems are as follows. First, the users interact and share their choices over the user interface. Second, both implicit and explicit data gets collected from the users. Third, the collected data goes through the recommendation engine. The recommendation engine can also be divided into four steps: preprocessing, learning, evaluation, and prediction. Lastly, the top predicted recommendations are forwarded to the users.

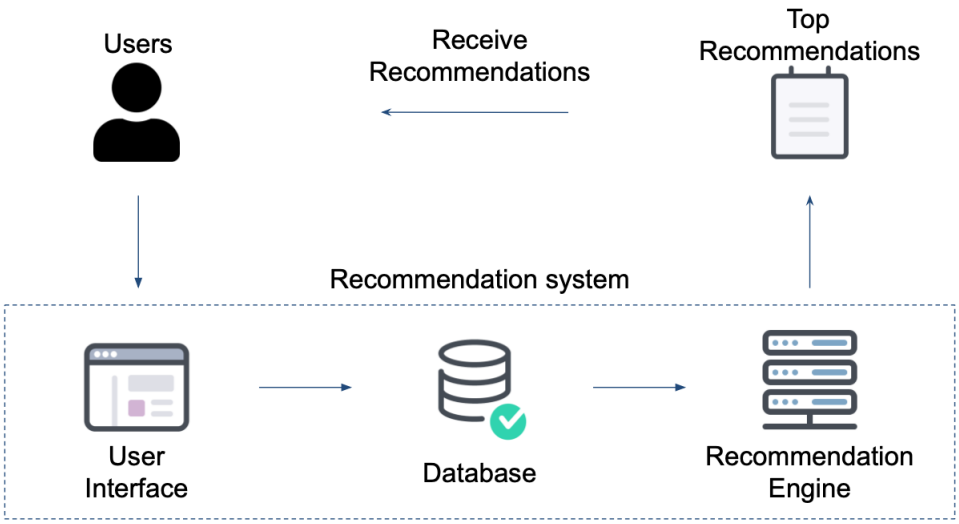

Figure 1.1: Steps of Recommendation Systems

Traditional Recommendation

There are three types of traditional recommendation system approaches (1) content-based, (2) collaborative filtering, (3) hybrid. Content-based recommends products based on users' preference on items. Collaborative filtering recommends products based on similar users' tastes. Hybrid is a mixture of both. Recent work of Matrix Factorization (MF) [3] uncovers latent space of users' and item embedding, which is a type of content-based. Memory-based models use collaborative filtering to find similar users' tastes. Hybrid model such as DeepFM [5] is a mixture of matrix factorization [3] and DNN models.

Sequential Recommendation

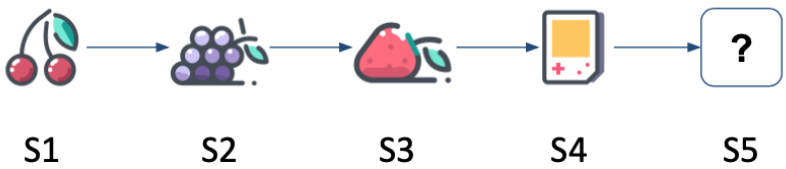

Figure 1.2: Item Sequence

In the real world, the user-item transaction happens in sequential matters rather than unordered matters. Sequential information is critical to incorporate into the recommender system. Figure 1.1 shows that the user-item sequence S1 to S4 goes from cherry, grapes, strawberry, and Game Boy. The task is to predict the next item, S5. The next predicted item, S5, will be an item similar to fruit in a traditional recommendation system, like an orange. However, since the last viewed item is a Game Boy, it is more likely that the user will buy something similar to a gaming device. Taking sequential information will have a better user experience in a recommendation.

Sequential recommendation predicts users' following items by modeling sequential information over user-item interactions. The traditional recommendation is based on users' preferences and can fail to capture the current and recent preferences. To enhance the recommendation system, it needs to incorporate time information in the recommender system.

1.1.3 Motivation

Building a recommendation system is not easy, especially a sequential recommendation. Though sequential-based models have shown success in terms of accuracy, these solutions suffer from the following problems.

First, there is no unified solution between traditional and sequential recommendation systems. Traditional and sequential recommendations are two different types. The traditional recommendation is user-item based, and sequential is sequence behavior-based. The two settings are very different. The differences include problem formulation, data attributes, evaluation, metrics, etc. Even though the settings are very different, they share some valuable methods that can be used, such as neural models, loss functions, negative sampling, etc. Two settings have their advantages, but currently, no approach unifies both problems. There is no fine-grained neural model to tackle both recommendation problems. So the first challenge is to unify two settings.

Second, the recommendation tasks evolve. In research fields like machine learning, many different methods are proposed constantly. For example, the evolution from Markov Chain [6] to RNN models took only a couple of years. Within RNN models, every year, researchers propose new tricks to the models, and the state of the art models change. Also, the data evolves. When

the data evolves, a great effort for tuning is needed. For example, OpenRec [7] tries to address the issue of the recommendation evolution problem by utilizing computational graphs and well-defined interfaces. However, it fails to handle new data and evolving tasks. Therefore, the second challenge is to build an adaptable recommender system.

Third, machine learning is a complex topic. For beginners, they need to learn both ML concepts and ML programming as well. Also, unlike other programming, ML requires an intuition that will leverage to solve a problem. Without ML intuition and knowledge, tackling the recommendation system is a complex problem. Therefore, the third challenge is to build an approachable system.

1.2 Contributions

This research primarily addresses problems regarding automated sequential recommendation systems.

(1) Does AutoSRec outperform state-of-the-art models? (2) How accurate are the sequential task results? (3) What is the influence of the model search space with the AutoSRec components? (4) How effectively does the hyperparameter optimization affect accuracy?

2. AUTOMATED SEQUENTIAL RECOMMENDATION

To address the above questions, we introduce an automated sequential recommendation called "AutoSRec."

2.1 Introduction

Sequential recommendation is a complex application scenario in building recommendation systems. For example, the sequential recommendation system needs to learn the user-item interaction and the sequential pattern—a sole sequential model limits to capture these two goals. Moreover, the settings are very different for each sequential scenario. Koran [8] proposed a time-aware dynamic model with collaborative filtering to find patterns between items and users. However, Koran [8] model is limited to rating predictions; many of the sequential recommendations extend to click-through rate or other data attributes.

Markov Chain models a widely used approach. FPMC [6] proposed a factorized personalized Markov Chain method by using both sequential information and personalized item information. Even though this method outperforms traditional recommendation methods, it only counts previous short orders. Higher-order sequential dependencies are hard to model by Markov Chain.

To address the higher-order sequential interactions, another method that arose is based on latent representation. Ruining He [1], proposed a Translation-Based Recommendation. It embeds items and users as a point in a latent space. It can deal with higher-order interactions and large sequences by representing a latent space due to its simple form.

One of the improved sequential methods is the Recurrent Neural Network (RNN) models. Hidasi [9] proposed a GRU based recommendation. GRU can capture both short-term and long-term dependencies by nature. By gated architecture, it can control the information and keep previous information for a long time.

Within the above examples, many sequential approaches lack short-term or long-term dependencies. To solve this, researchers use the help of traditional recommendation models such as

matrix factorization and collaborative filtering. FPMC [6] uses a combination of matrix factorization and Makov Chain to capture both long-2term user-taste, and short sequential preference. Caser [10] uses CNN-method by capture higher-order Markov Chain. Many approaches use a mixture of both traditional and sequential to improve on each shortcoming. AutoSRec aims to have a unified search space of both traditional and sequential recommendations.

2.2 Problem Statement

Given a set of items $S_n : (S_1^u, S_2^u, ..., S_{|S^u|}^u)$ and the users' rating is given as $R_n : (R_1^u, R_2^u, ..., R_{|R^u|}^u)$. n is denoted as the randomized order and u is denoted as user-ID. We denote the next K items as $S_k : (S_{k-1}^u, S_{k-2}^u, ..., S_k^u)$. We denote the test ratings as $R_k : (R_{k-1}^u, R_{k-2}^u, ..., R_k^u)$. Also, $S_k \subset \mathrm{S}_n$ and $R_k \subset \mathrm{R}_n$

Based on the notations above, we define our problem as follows: A set of sequence items S_n and a corresponding set of sequence ratings R_n are given. The goal is to score the predicted rating R_k as the highest. Then, the top scores are the next k items S_k.

2.3 AutoSRec

Sequential recommendation is a complex application scenario in building recommendation systems. For example, the sequential recommendation system needs to learn the user-item interaction and the sequential pattern—a sole sequential model limits to capture those two goals. Moreover, the settings are very different for each sequential scenario. Koran [8] proposed a time-aware dynamic model with collaborative to find patterns between items and users. However, Koran [8] model is limited to rating predictions; many of the sequential recommendations extend to click-through rate or other data attributes.

AutoSRec uses automated machine learning, AutoML, to build a universal search space for a sequential recommendation. Sequential and traditional models are involved in the search space to expand both user-item interaction and sequential patterns. For example, AutoSRec builds a pipeline of two orders: LSTM and MLP. Moreover, each useful trick has been modified to be used in AutoSRec. For example, AutoSRec uses negative sampling to rank the items.

Unlike other recommendation systems, AutoSRec provides the end-to-end solution from prepossessing, model selection, hyperparameter tuning to the next basket recommendation. To solve motivations from section 1.1.3 above, AutoSRec aims to provide a unified search space for traditional and sequential recommendations to solve the next basket recommendations.

2.4 AutoSRec Framework Overview

AutoSRec mainly uses TensorFlow and Keras as the framework. These two are some of the most dominant machine learning frameworks out there. Then, we use Autokeras [11] and Keras Tuner to expand ML to an AutoML. With these frameworks, we can build an automated sequential recommendation, AutoSRec.

2.5 AutoSRec Architecture Overview

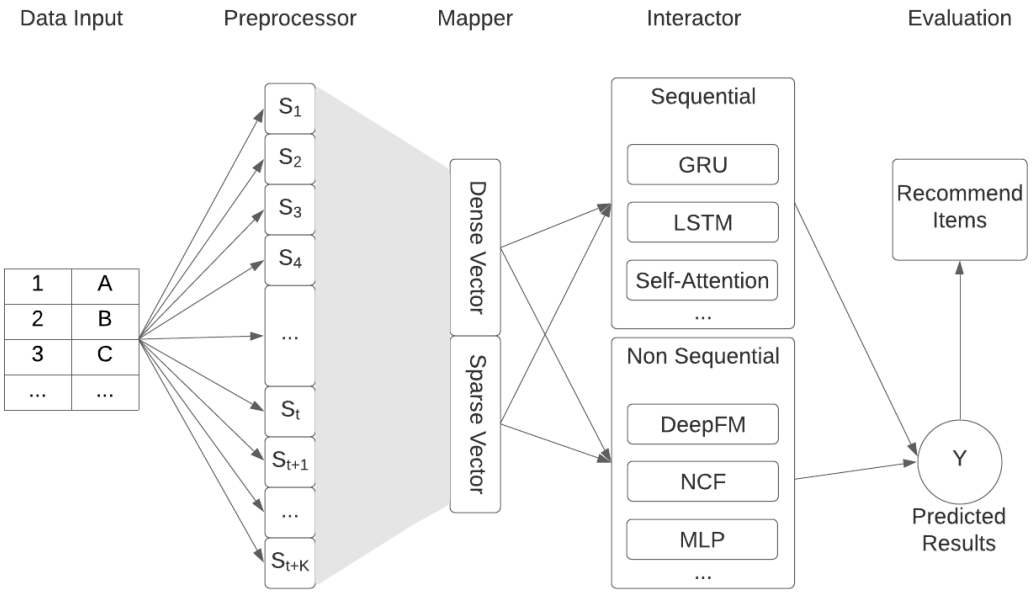

Figure 2.1: Architecture of AutoSRec

As shown in Figure 2.1, AutoSRec can be summarized into four components: Preprocessor, Mapper, Interactor, and Evaluation. Mapper and Interactor are special AutoKeras [11] blocks

where it can be put together as a recommendation's model search space. Preprocessor and Evaluation is not a specialized block, but its task is to adapt among different types of sequential data.

- Preprocessor converts recommendation data to fit the AutoSRec inputs. User, item, and time attributes are converted into a sequential matter for the model to learn. Some features include data input and output, time padding, time-shifting, outlier remover, etc. Many of the recommender's preprocessing techniques and sequential techniques are involved in this step.

- Mapper is a special block that involves the search space. Data features are converted to a latent factor, so the model can do a better analysis and relate. The numerical features are mapped as embeddings. The categorical features are transformed and mapped as embeddings[4].

- Interactor is a special block that involves the search space. In this component, there are 6 prestructed interactions including, MLPInteraction, NCFInteraction [12], FMInteraction [5], LSTMInteraction [13], GRUInteraction [9], SelfAttentionInteraction. Furthermore, we separated these interactions into non-sequntial, and sequential blocks. HyperInterction1 includes non-sequential blocks which are MLP, NCF, and DeepFM. HyperInteractions2 includes LSTM [13], GRU [9], and Self-Attention [14]. All of these interactions are in the search space meaning that they have tunable hyperprameters.

- Evaluation component measures the accuracy of the result. Unlike general recommendation, where it evaluates based upon mse, logloss, or others, sequential recommendation evaluates upon hit rate (HR), mean reciprocal rank (MRR), and normalized discounted cumulative gain (NDCG). These metrics are essential in evaluating the sequential models. These metrics are not currently available in the Keras framework. We involved these metrics and other evaluation techniques in the Evaluation stage.

2.6 AutoSRec Models

AutoSRec has 8 models including AutoSRec's special model: MLP, NCF [12], DeepFM [5], GRU [9], SAS [14], and LSTM [15]. These models can be built upon using the component of

AutoSRec. For example, DeepFM [5] is built upon by connecting dense/sparse mapper, FMInteraction and MLPInteraction. Similarly, other models can be built with AutoSRec's current components.

3. METHODOLOGY: OUR APPROACH

Table 3.1: Table I: Notation

Notation	Description
U, I, R	user, item, score set
S^u	historical interaction sequence for an item $u : (S_1^u, S_2^u, ..., S_{\|S^u\|}^u)$
R^u	historical interaction sequence for a score $u : (R_1^u, R_2^u, ..., R_{\|R^u\|}^u)$
W^u	unseen items $u : (W_1^u, W_2^u, ..., W_{\|W^u\|}^u)$
$d \in$N	latent vector dimensionality
$n \in$N	maximum sequence length
$k \in$N	maximum number of prediction
E	input embedding matrix
P	positional embedding matrix
M	Models
Ω	search space

3.1 Sequential Recommendation

A sequential recommendation's problem formulation is different from a traditional recommendation. Let U be a set of users, I be a set of items, and R be a set of scores on items. Then, the users' action sequence is given as $u : (S_1^u, S_2^u, ..., S_{|S^u|}^u)$ and the users' rating sequence is given as $u : (R_1^u, R_2^u, ..., R_{|R^u|}^u)$. The model's input is similar to Figure 1.1. The models input is $u : (S_1^u, S_2^u, ..., S_{|S^u|-k}^u)$ and the expected output sequence would be $u : (S_{|S^u|-k+1}^u, S_{|S^u|-k+2}^u, ..., S_{|S^u|}^u)$. The object of sequential recommendation is to predict the next basket of items that users will interact with, given a historical set of items and scores.

3.2 AutoSRec

AutoSRec block order is set: preprocessor, mapper, interactor, and evaluation. The user can customize within the blocks. The details of each block are as follows:

3.3 AutoSRec Preprocessor

Most of the recommendation is a tabular dataset with a timestamp. Preprocessor converts tabular dataset to item and score sequence: $u : (S_1^u, S_2^u, ..., S_{|S^u|}^u)$ and $u : (R_1^u, R_2^u, ..., R_{|R^u|}^u)$. For the training dataset, all the user-item sequence processed to be a length of 50: $u : (S_1^u, S_2^u, ..., S_{50}^u)$ and $u : (R_1^u, R_2^u, ..., R_{50}^u)$. AutoSRec collects the last five viewed items for the test dataset, $u : (S_{51}^u, S_{52}^u, ..., S_{55}^u)$, and combines them with 95 negative unseen items, $u : (W_1^u, W_2^u, ..., W_{95}^u)$. Then we have a test set of randomly shuffled 100 items: $u : (S_{51}^u, S_{52}^u, ..., W_{95}^u)$. We do not convert the corresponding scores on test data.

3.4 AutoSRec Mapper

We have three different types of embedding layers: dense embedding, sparse embedding, and positional embedding. Dense embedding handles numerical data types, sparse embedding handles categorical data types, and positional embedding handles attention modules.

Dense Embedding

Numerical features are dense so that it is better to represent them in a high dimensional space.

$$E_i^d = V_i^d x_i^d \tag{3.1}$$

V_i is the embedding matrix for numerical field i, and x_i is the numerical data for field i, and e_i is the output of dense embedding.

Saprse Embedding

Categorical features are very sparse so that it is better to represent them in a low dimensional space.

$$E_i^s = V_i^s x_i^s \tag{3.2}$$

V_i is the embedding matrix for categorical field i, x_i is the categorical data for field i, and E_i is the output of sparse embedding.

3.5 AutoSRec Interactor

AutoSRec interactors consists of two modules which are called non-sequential and sequential hyperinteractions. Non-sequential hyperinteraction consists of Multilayer Perceptron (MLP), Neural Collaborative Filter (NCF) [12], and DeepFM [5]. The Sequential Model consists of LSTM [13], GRU [9], and Self-Attention [14].

Seqential HyperInteraction

The purpose of sequential hyper-interaction is to learn the sequential trend in users and item embedding.

$$M_1, M_2, M_3 \in \Omega_{seq} \tag{3.3}$$

$$O = \Omega_{seq} * \widetilde{E} \tag{3.4}$$

Let Ω_{seq} be the hyperinteractions in the search space. Then, $\Omega_{seq} \in M_1, M_2, M_3$, $M1$, $M2$, and $M3$ are respectively, LSTM [13], GRU [9], and Self-Attention [14]. For each iterating trials, best models in Ω_{seq} is searched. Also, the depth of Ω_{seq} can be tuned. In this stage, both models search and hyperparameter optimization is used.

Non-Sequential HyperInteraction

The purpose of non-sequential hyperinteraction is to learn implicit and explicit data interactions.

$$M_1, M_2, M_3 \in \Omega_{non-seq} \tag{3.5}$$

$$\widetilde{O} = \Omega_{non-seq} * O \tag{3.6}$$

Let $\Omega_{non-seq}$ be the hyperinteractions in the search space. Then, $\Omega_{non-seq} \in M_1, M_2, M_3$, $M1$, $M2$, and $M3$ are respectively, MLP, NCF [12], and DeepFM [5]. For each iterating trial, the best model in $\Omega_{non-seq}$ is searched. Also, the depth of $\Omega_{non-seq}$ can be tuned. This $\Omega_{non-seq}$ is placed either before or after the sequence models. For our case, the best accuracy was after the sequence

search space.

3.6 AutoSRec Evaluation

The loss function of AutoSRec in the sequential scenario is a combination of two loss functions: Regression and Classification.

$$Loss_{final} = (1 - \delta)Loss_{mse} + (\delta)Loss_{log}$$

The delta in our case is 0.5.

Regression

The regression measures the mean squared error between the true scores and the predicted scores.

$$Loss_{mse} = \frac{1}{n}\sum_{i=1}^{n}(y - \widetilde{y}_i)^2 \tag{3.8}$$

Classification

The classification measures the log loss between the true items and the predicted items.

$$Loss_{log} = -\sum_{i=1}^{c} log(q_i), q_i = probability \tag{3.9}$$

AutoSRec uses two loss functions in order to deeply learn the sequence change in both scores and item trends to minimize the trend loss.

Prediction

After minimizing both loss, the score on test set, $u : (S^u_{51}, S^u_{52}, ..., W^u_{95})$, is predicted. The top five score is the next basket recommendation.

3.7 An Example of AutoSRec Pipeline

With multiple experiments and results, we found that this type of pipeline gave us the best results. The pipeline steps are as follows:

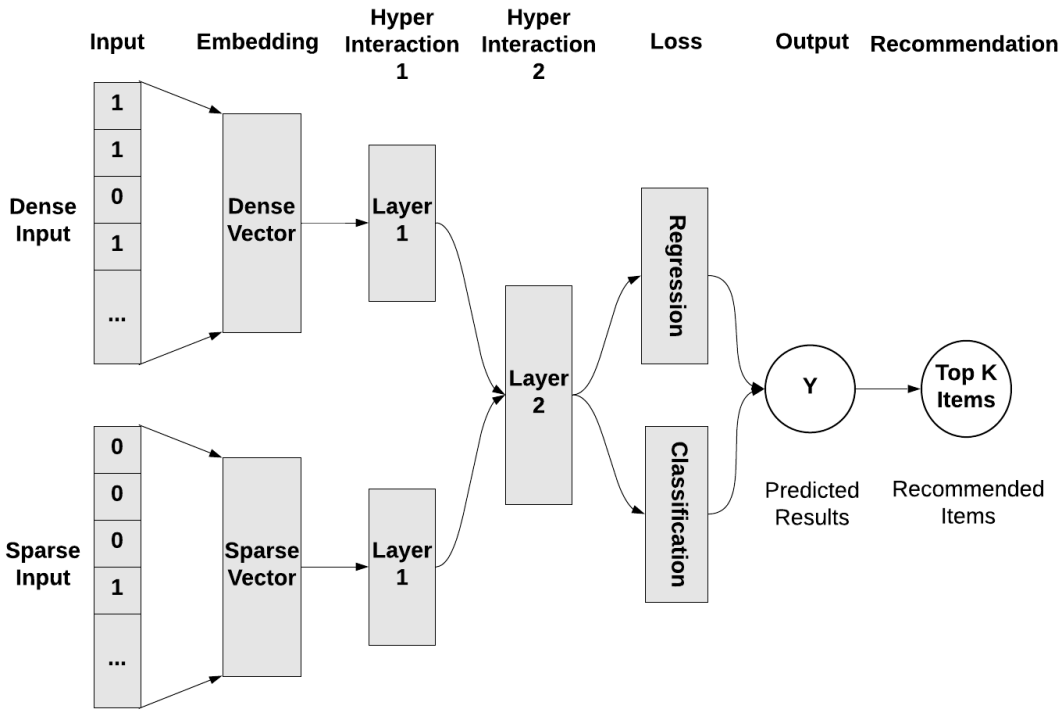

Figure 3.1: Architecture of AutoSRec

The data has divided into two inputs: dense and sparse inputs. Dense inputs include user age, item release date, etc. Sparse inputs include user location, item genre, etc. These two inputs are processed in their method since they have different properties. For example, the dense layer is embedding with MLP, and sparse features are mapped into numerical vectors via embedding lookup. Then, sequential hyper-interaction is used to extract the historical pattern in each dense and sparse input. Fourth, non-sequential hyper-interaction is used to learn user-item interaction beyond the sequential information. Then, we calculate the two losses: regression and classification. Regression minimizes the loss of scores on items, and classification minimizes the categorical cross-entropy on item ID. After minimizing the loss, we only take the regression loss to find the highest score on the item. With experiments, adding classification loss to the evaluation decreases the accuracy. Not just AutoSRec, but many of the sequential recommendations only uses regression loss to find the next item. Then, the highest top K score items are the recommended items, and this is the next basket item user will likely watch.

4. EXPERIMENTS

We evaluate the AutoSRec model with real-world datasets to answer our research questions.

RQ1: Does AutoSRec outperform state-of-the-art models?

RQ2: How accurate are the sequential task results?

RQ3: What is the influence of the model search space with or without the AutoSRec components?

RQ4: How effectively does hyperparameter optimization affect accuracy?

4.1 Data Descriptions

Three popular datasets from the real-world application are used to evaluate our model. We used three very different item categories: movie, sports, and beauty.

- MovieLens: This is widely used data for evaluation in recommendation system. MovieLens 1M is used for our dataset.
- Amazon: Two versions of Amazon datasets are used, which are sports and beauty. Amazon datasets are very sparse.

We truncated some of the shortest and the longest user sequences to remove bias in the data. We discard users and items with fewer than 10. The data statistics are shown in Table 4.1. MovieLens is the densest data. Two Amazon data have similar densities, but the two data categories are very different.

Table 4.1: Table I: Dataset Description

Dataset	Users	Items	Interactions	Avg actions per user
MovieLens	202	3,189	66,859	330
Sports	459	12,676	20,147	44
Beauty	1,078	27,896	55,375	51

4.2 Evaluation Metrics

For evaluation metrics, we used three types: hit rate (HR), mean reciprocal rate (MRR), and normalized discounted cumulative gain (NDCG).

$$HR@K = \frac{1}{M} \sum_{u \in U} (R_u < K) \tag{4.1}$$

Figure 4.1 is the HR measurement. HR measures the accuracy of recommended prediction. K is the top predicted items. M is the total number of users, U. R_u is the predicted item rank. If R_u is among top K, then the function returns a 1, otherwise 0.

$$MRR@K = \frac{1}{M} \sum_{u \in U} max(\frac{1}{R_u < K}) \tag{4.2}$$

Figure 4.2 is the MRR measurement. MRR measures the top position of the ranked item. The highest rank can be 1, and the lowest rank can be $\frac{1}{K}$. The function returns the position of the highest rank.

$$NDCG@K = \frac{1}{M} \sum_{u \in U} \frac{1}{R_u < K} \tag{4.3}$$

Figure 4.3 is the NDCG measurement. The highest rank can be 1, and the lowest rank can be $\frac{1}{K}$. The function returns the sum of all rank positions.

4.3 Compared Models

We compare AutoSRec models with six different baselines.

1. **Pop.** This approach ranks the item based on the popularity of items.

2. **Random.** This approach ranks the item based on randomly selected items.

3. **NCF [12].** This approach ranks the item based on user-user interaction.

4. **DeepFM [5].** This approach ranks the item based on high-low feature interactions.

5. **SAS [14].** This approach ranks the item based on self-attention mechanisms.

6. **GRU4Rec [9].** This approach ranks the item based on temporal sequence.

4.4 Implementation Details

All baselines and AutoSRec model is implemented in python with Tensorflow. All experiments were conducted on GeForce RTX 2080 GPU. In order to avoid biases and randomness in the dataset, we dropped all users in less than ten sequence lengths. For each user, we negatively sample unseen items and add K next items. Then, we rank these predicted items and compare them with ground truth to evaluate.

4.5 Performance Comparison

Table 4.2: Table I: Performance Accuracy

Dataset	Metrics	Pop	Random	NCF	DeepFM	SAS	GRU4Rec	AutoSRec
MovieLens	HR@5	0.0658	0.0684	0.0849	0.1176	0.2063	0.3743	**0.4110**
	MRR@5	0.1581	0.1642	0.1646	0.1990	0.2630	0.3099	**0.4356**
	NDCG@5	0.0906	0.0828	0.0739	0.1259	0.1763	**0.3354**	0.3013
Sports	HR@5	0.2312	0.2344	0.3750	0.3844	0.4147	0.6083	**0.6219**
	MRR@5	0.4431	0.4464	0.5345	0.5583	0.5184	0.5583	**0.5706**
	NDCG@5	0.2519	0.1781	0.3335	0.3424	0.3883	**0.5947**	0.5646
Beauty	HR@5	0.1794	0.1864	0.3436	0.4103	0.4428	0.5382	**0.5973**
	MRR@5	0.3379	0.3590	0.3423	0.4051	0.4231	0.5232	**0.5539**
	NDCG@5	0.1973	0.1427	0.3120	0.3790	0.4272	**0.4949**	0.4391

Table 4.2 presents the recommendation performance of all methods on three datasets **RQ1**. The sequential models (SAS, GRU4Rec) perform better than other models in the next basket recommendation. The AutoRec is a combination of traditional and sequential models. The AutoSRec outperforms all three datasets in HR and MRR. Movielens is a challenging dataset; pop and random models do not perform well compared to the other two datasets. However, AutoSRec achieves a 9.8 percentage increase in HR compared to the GRU4Rec. For all NDCGs, AutoSRec does not

achieve the best score. Presumably, GRU4Rec only utilizes a sequential model, and AutoSRec uses both sequential and traditional models. In AutoSRec, the traditional model can capture better user-item scores, but it loses the sequential position information.

4.6 Detailed Results

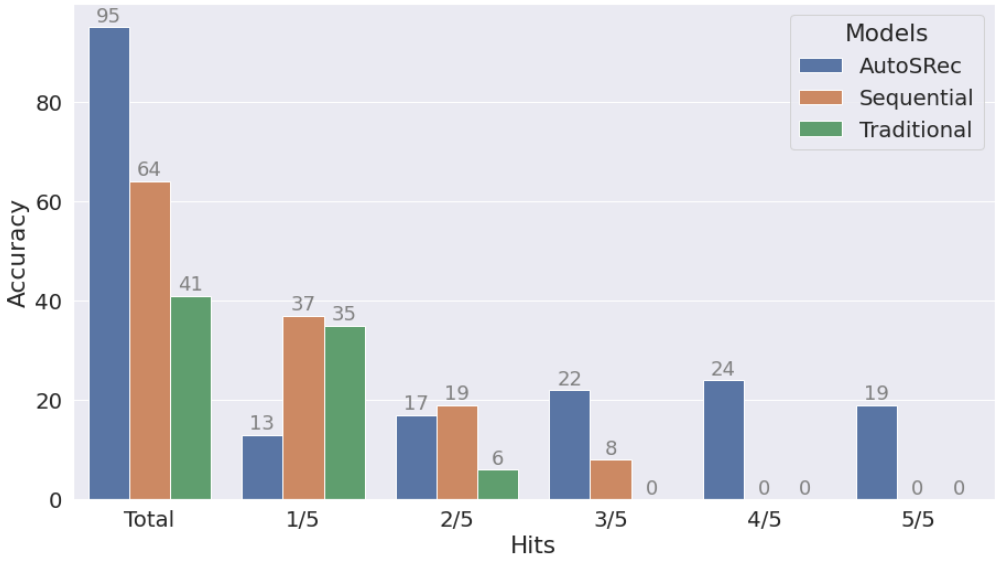

Figure 4.1: Detailed Results

In sequential elements, the goal is to capture users' trends in taste. Higher HR might not mean that the trend is accurate. To further analyze the quality of our result and to answer **RQ2**, we conducted a detailed analysis of our result. In figure 4.1, we divide into three categories of model: AutoSRec, sequential, and traditional. The x-axis is the category of the top 5. The y-axis is the number of users that is a hit. For example, AutoSRec's 3 out of 5 has 22 hits. This means that there are 22 hits that got 3 out of 5 correct.

AutoSRec has the most total HRs, the sequential model's second, and the traditional model is third. AutoSRec hits of baskets are overly distributed among all five. As it goes from 1 out 5 to 4 out of 5, the number of hits increases. Then, at 5 out of 5, the number of hits decreased. Presumably, as the next basket increases, it gets more challenging for the model to capture the

trend. Compared to other approaches, only using sequential model does not capture more than 3 out of 5. We find that AutoSRec search space leads to a better performance.

4.7 Ablation Study

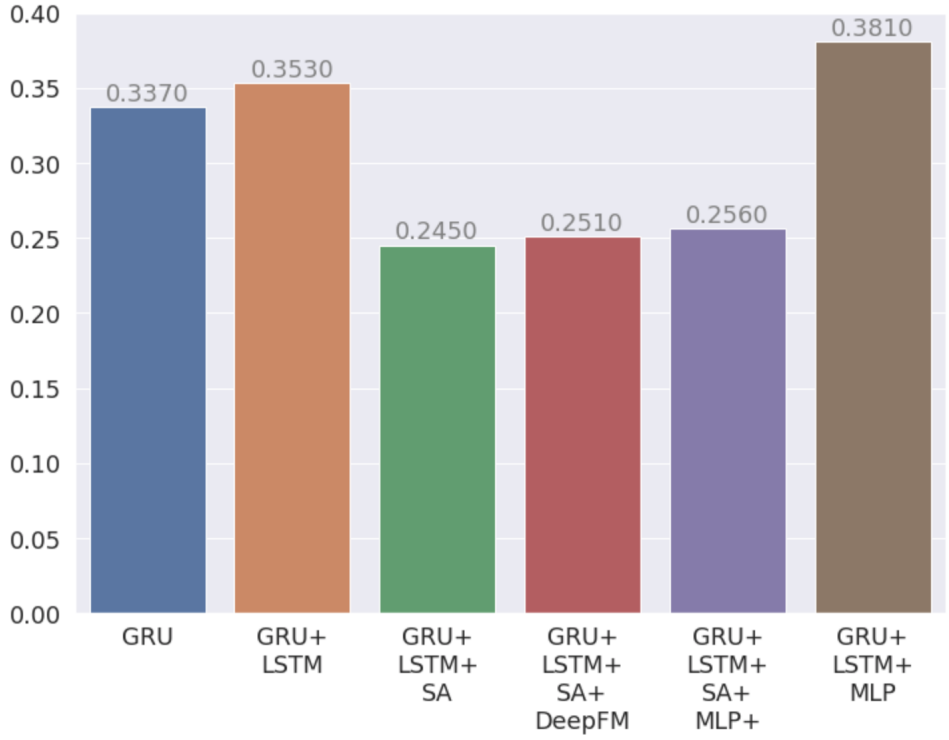

Figure 4.2: Ablation Study

Many factors can affect our model. One of the most significant factors is the search space. To answer **RQ3**, we conducted an ablation study removing components in the search space.

1. GRU is our default baseline because that has the highest accuracy among baselines.

2. Adding LSTM into the search space increased the accuracy.

3. Adding Self-Attention into the search space decreased the accuracy. This is because Self-Attention, in our case, does not work effectively as others.

4. To further analyze search space, we start adding traditional models. The accuracy increased as we added a DeepFM.

5. Adding MLP to the search space increased accuracy

6. Then created our search space with only the best performing models. We find that the GRU, LSTM, and MLP gave the best accuracy.

The best accuracy was from (6). Combining the best models in the search space increase the AutoSRec's accuracy compared with only the GRU model. AutoSRec is able to find a better model pipeline compared to only one GRU model (1).

4.8 Hyperparameter Study

Table 4.3: Table I: Hyperparameter

Model	Fixed	Non-Fixed
AutoSRec	0.2869	0.4110

To answer **RQ4**, we conducted a hyperparameter test. From Table 4.3, Fixed means that we used predefined research models hyperparameters. Non-Fixed means that we gave leverage to the fixed hyperparameters number for AutoSRec to optimize. The result shows that AutoSRec's hyperparameter optimization increases accuracy significantly.

5. CONCLUSIONS

In this paper, we proposed AutoSRec, an Automated Sequential Recommendation, to address current problems of sequential recommendations. First, AutoSRec unifies the sequential recommendation tasks. Second, AutoSRec adapts different data tasks, models, and structures. Third, AutoSRec is built upon a function API coding style where it is intuitive for programming beginners to start.

In conclusion, the AutoSRec achieves three goals; automation, adaptability, and usability. For the results, AutoSRec is indeed a competitive model among the sequential recommendation society; it achieves a state-of-the-art model. The model search, which is the main component of AutoSRec, increases the accuracy. Hyperparameter optimization does find the optimal parameters in models. In the end, AutoSRec hopes to attract many users who are interested in this field.

For future studies, self-supervised learning opens up an interesting topic for recommendation systems. This technique enables experiments on data sets that don't have labels. No label data sets are a more real-world scenario because getting a label is very costly. Therefore, we want further to study more difficult real-world datasets with this technique.

REFERENCES

[1] R. He, W.-C. Kang, and J. McAuley, "Translation-based recommendation," in *Proceedings of the eleventh ACM conference on recommender systems*, pp. 161–169, 2017.

[2] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[5] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[6] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*, pp. 811–820, 2010.

[7] Ylongqi, "ylongqi/openrec."

[8] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 447–456, 2009.

[9] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[10] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 565–573, 2018.

[11] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, 2019.

[12] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 197–206, 2018.

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.