Adapting Sequential Recommender Systems to Predict User Behaviour in Future Temporal Windows

Umer Gupta

Master of Science

Data Science

School of Informatics

University of Edinburgh

2023

Abstract

Sequential Recommendation Systems (SRSs) focus on predicting the very next item [14, 32]. [24] suggests focusing on predicting items that a user will interact with in a future temporal window instead. Novel learning ideas that train on future windows are required to optimise for this new task. This is window-based learning. Any research in this domain is restricted to [24] in our knowledge. Hence, we expand on window-based learning ideas suggested in the paper. Learning ideas proposed in [24] as well as new learning ideas contributed by this body of work are evaluated. The findings show that window-based learning improves predictions over the future window. This means window-based learning objectives are a viable option in use cases where predictions over long time periods are required. For example, if a fashion brand wants to catalogue for the following month. Moreover, we look at models that combine sequential next item learning ideas, as in the SASRec [14], with window-based learning. These models yield the best results. Our best combined model improves on the next item learner by 17.4% and 25% on the Recall@10 and NDCG@10 Top-N metrics respectively. Furthermore, window-based and combined models show a better performance than models trained daily on the next item learning objective over a 14-day window even when trained only once before the window. This opens up the possibility of SRSs to be updated in window-sized cycles instead of daily in deployment scenarios. The sparser update cycles mean less resources expended.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Umer Gupta*)

Acknowledgements

I would like to thank Amazon for the amazing opportunity to work on this current and relevant industry-based project focused on advancing Recommender Systems. Special thanks to the team of supervisors at Amazon: David Wardrope, Timos Korres, Michael la Grange and Thomas Baumer. I have been fortunate to work under their guidance.

I would also like to thank Iain Murray, my internal supervisor on the project, for his prompt and thorough feedback throughout the duration of the project. Having joined the Master's program late, I would not have got this far but for his continued support.

Lastly, I would like to thank my peers - Yunlong, Kyriakos and Mari - on the Future Recommendations project for sharing this journey with me and making it a memorable one.

Table of Contents

Chapter 1

Introduction

IT based services use Recommendation Systems (RSs) to avoid overloading users with information. RSs understand user preferences to filter through the noise and provide users with information that is useful to them. Traditional RSs operate on the entire history of user interactions and ignore the ordering of these interactions. The underlying assumption is that users have static preferences [6]. However, a user who historically likes console gaming is not likely to buy a "XBox" right after buying a "PS4" [32]. User preferences are actually dynamic. Sequential RSs then look at the historical *sequence* of user interactions to understand the user's *dynamic* preferences and predict future interactions [5, 6].

Conventional sequential recommendation systems (SRSs) are optimised to predict the very next interaction a user will make [5, 14]. This is what we refer to as the next item prediction task. Currently, most SRSs are trained for this task. For example, SASRec [14], a self-attention based sequential recommender, tries to predict the next item at every point in an historical user interaction sequence during training to optimise for this task. This is the sequential next item learning objective (or simply the next item learning objective in this work). SASRec achieves state-of-the-art (SOTA) performance on the next item prediction task.

Pancha et al. [24] instead propose a novel task: predict interactions users will make in a future temporal window beginning from their most recent interaction. They propose a model, Pinnerformer, optimised for this task by training on window-based learning objectives. Window-based learning objectives mimic temporal windows to learn on during training. In the paper, this model is trained once before making predictions over a 14-day future window. It achieves similar performance to models that are trained daily to make predictions on the following day over the same window [24].

This is a significant result. It shows optimising models to predict on future windows could allow us to train models far less frequently at a much lower cost in terms of performance. This makes window tasks lucrative because sparser model updates mean fewer resources expended in deployment scenarios [24]. Furthermore, it could be useful to predict what users might do further in the future. For example, if a fashion brand wants to pick its catalogue for the following month it would want to know the kind of products users are likely to interact with in the entire one month period.

To the best of our knowledge, the research produced at Pinterest [24] is the first and only piece of work on window-based predictions using sequential recommendation systems. The research is primarily aimed at improving the efficiency of their RS by utilising the window-based predictions. It does not try to develop a general overview of window-based prediction task itself. That is to say, the Pinnerformer was developed and optimised to improve performance on a specific and internal data regime, i.e., user interactions on Pinterest. Hence, there is a need for further experimentation on window-based methods and ideas to get a broader understanding of the concept and make stronger, more general statements about the same [6].

This project builds on the research produced at Pinterest. Experiments are conducted on the MovieLens dataset, which is commonly-used and open-source. We answer the following research questions :-

- **[RQ1]** *Can we create new learning algorithms to make better predictions on future windows?*

  The popular SASRec [14] model is extended to train on window-based learning objectives. We implement window-based learning objectives suggested by [24] as well as new variations contributed by this body of work. The hypothesis is that window-based learners will outperform the next item learner on the future window in terms of Top-N metrics. The results show that window-based learners in fact outperform the next item learner.

- **[RQ2]** *Are sequential learning and window-based learning related? If so, how can we exploit this relationship to squeeze out the optimal performance on the future window?*

  A major achievement of this research dissertation is combining sequential learning with window-based learning. It is hypothesised that the sequential next item learning objective and the window-based learning objective are related. The window next item learning objective, a combined learning model, performs better

than the window-based learners suggesting that there is in fact a relationship between the two learning ideas. We further investigate this through transfer learning (TL). The idea is to understand how well sequential knowledge acquired on next item learning transfers onto window-based learning. We hypothesise that the approach will improve the performance of window-based learners, thereby confirming the existence of an underlying relationship between the two ideas. This is, in fact, what we observe.

Further, we hypothesise that there is an optimal balance between the two learning ideas such that performance is maximised at this configuration. A hybrid model is used that can cycle between learning objectives to control the amount of focus on each of the learning ideas. We see that such an optimal configuration exists. In fact, this hybrid learner at the optimal combination gives the strongest performance out of all the different learning models.

- **[RQ3]** *What is the effect of incorporating temporal information?*

  Time features are added. Given that the window-based prediction task is motivated by temporal ideas, our hypothesis is that the performance of the models should improve. In general, time features do improve performance of the models.

- **[RQ4]** *Can using window-based learning motivate more efficient deployment-level RS architectures?*

  Similar to [24], we hypothesise that updating in window-sized cycles (every 14 days in our case) is a viable option because window-based and combined learners trained on these cycles are able remain competitive with next item learners trained daily. The results show that window-based and combined learners are not only competitive but outperform the next item learner trained daily. This motivates the use of sparser update cyles in deployment scenarios which in turn saves time and resources.

The rest of this paper is structured as explained here. Chapter 2 discusses the relevant literature and introduces the required background knowledge. Chapter 3 talks about the methodology adopted by us. Chapter 4 presents and analyses the results of the experiments. Finally, Chapter 5 gives the concluding remarks, highlighting the major contributions of this study, the limitations of our work and the future areas of research the work motivates.

Chapter 2

Related Work and Background

2.1 Related Work

Most early research surrounding RSs was focused on Collaborative Filtering (CF) ideas which gained popularity after the Netflix Prize competition [17, 18, 21, 29]. The popular Matrix Factorization (MF) is an example of this class of methods [17, 28]. It projects items and users to a common latent space which is then used to directly predict a users preference on an item by utilising the item and user vectors [17, 18].

The methods in this paper are focused on the sequential recommendation which uses ordered histories of user interactions. Early sequential recommenders employed Markov Chains (MCs)[32]. Works include [30] that contextualises the task as a Markov Decision Process (MDP) and [27] that uses MC and MF ideas in combination. [8] and [9] leverage higher order MCs [18].

Soon, more powerful Deep Learning ideas were being used for SRSs. Because of their adeptness in dealing with serial information, Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs) and Long Short-term Memory (LSTM) quickly gained wide appeal [3, 10, 20, 25, 37]. These recurrent architectures are able to take a vector of the past user interactions and learn representations that describe user preferences [32]. Besides RNNs and its variants (GRUs and LSTMs), other interesting architectures like Convolutional Neural Networks (CNNs) [33] and Memory Networks [1, 12] also got adopted [32].

However, recently, transformer based models have completely dominated SRSs [6]. Given the impressive performance of transformers on machine translation [34], which is also a task focused on sequential learning, Kang and McAuley were inspired to create a purely attention-based model, SASRec [14], for SRSs.

The attention mechanism retrieves learned representations that describe user preferences from a vector of past user interactions like in RNNs; however, it does away with the need to explicitly encode sequential ideas in the model structure. Instead it integrates sequential information through simple positional embeddings added to the feature vector [31, 34]. This circumvents bottlenecks like vanishing gradients inherently present in RNN type networks [6, 11]. The SASRec achieved SOTA results on the MovieLens 1 Million and various Amazon datasets [14, 32].

Following this, attention-based SRS models became common. BERT4Rec [32] trains on the bidirectional cloze task instead of the unidirectional objective of SASRec. The Time Interval aware SASRec (TiSASRec) [19], includes relative temporal information. [24, 38] are made context-aware by including item information such as image, caption reviews, ratings, etc [6]. [35, 39] improve quality of learned user embeddings through self-supervised learning [6].

All of the SRS models mentioned above are optimised for the next item task. The team at Pinterest in a paper as recent as 2021 propose the idea of window prediction instead (See Section 2.2.1) [6]. Their model, the Pinnerformer [24], incorporates a time and context aware transformer-based architecture that uses two novel window-based learning objectives to optimise for the window prediction task instead of the next item task (See 2.2.3). Their work shows that these new window-based learners can compete with next item learners on future windows even when they are trained only once before the window while next item learners are trained daily over the window.

2.2 Background

2.2.1 Recommendation Tasks

This research aims to understand if novel window-based and combined learning approaches can produce learned representations that are valid over longer future time periods. We call these longer time periods windows. In this section, we introduce the window prediction task.

2.2.1.1 The Classical Next Item Prediction Task

Conventional SRSs process a history of past user interactions to predict the next future interaction. An interaction is defined by the type of action the user takes and the item that is acted on. For example, if a user “views socks” on a fashion e-commerce website,

then "view" is the action and "socks" is the item and both together compose a user interaction. For the purpose of this study we restrict ourselves to a single action at a time. That is, a history of interactions for a user can be summarised solely by a sequence of items. For example, if we only consider user "views" on the e-commerce website. If the user views "socks" followed by "shoes", the user interaction sequence can be summarised by ("socks", "shoes") as the action remains constant, i.e., "view" [6].

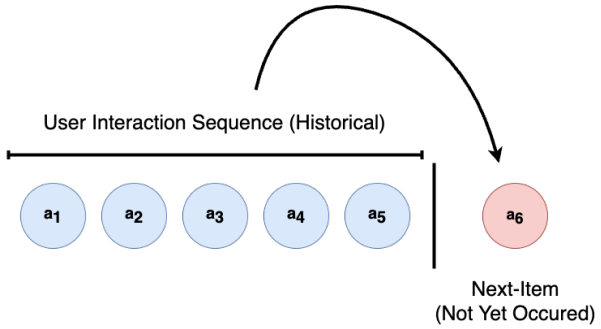


Figure 2.1: Window Prediction Task

To formalise this, let $U = \{u_1, u_2, ..., u_j\}$ be the set of all users and $I = \{i_1, i_2, ..., i_k\}$ be the set of all possible items that a user can interact with. Let $S' = (a_1, a_2, ..., a_m)$ be the history of interactions made by some user such that $a_i \in I$. Then, next item sequential recommendation task predicts the following distribution,

$$\mathrm{P}(a_{m+1}|S')$$

That is, the probability distribution of being the next item in the user interaction sequence across all $i \in I$.

2.2.1.2 Introducing the Window Prediction Task

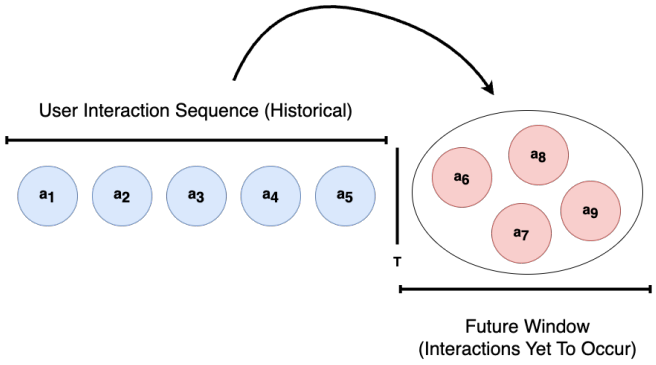


Figure 2.2: Window Prediction Task

The window-based prediction task, or window prediction task for short, extends the above framework and tries to predict the probability that an interaction will occur in a time period in the future. We are no more focused on the very next interaction but all interactions in a defined temporal period. Consider S' as before. Further, let's assume that this sequence encodes actions up till some time T. Then, the new task is to predict the following distribution,

$$\mathrm{P}(a \in W | S^{'})$$

where W is a defined time window starting at T. It is the set of all interactions made by the user starting from the time of their last interaction till some defined temporal distance in the future. The task is only interested in the probability of an item belonging to the window, not in the ordering or timestamps for these items (See Fig. 2.2).

2.2.2 SASRec

The SASRec [14] is a sequential recommendation model that adapts the transformer architecture to make recommendations (See Fig. 2.3). It forms the basis of all models used in this study. Sequential inputs are processed and transformed into relevant learned embeddings as described below for all models.

2.2.2.1 Architecture

The SASRec truncates or pads each user interaction sequence S' to create input sequences $S = (a_{m-n}, a_{m-n+1}, ..., a_m)$ of consistent length n. The user information is not explicitly encoded but assumed to be implicitly represented by the input sequence. This sequence is passed through stacked transformer blocks to extract information about item relationships in an hierarchical fashion. First, each element of the input sequence S is embedded to give $E_u = (e_1, e_2, ...e_n)$. Each $e_k = M_{a_k} + P_k$ where $M \in \mathbb{R}^{|I| \times d}$ is the item embedding matrix and $P \in \mathbb{R}^{n \times d}$ is the positional embedding matrix. Both are learned. d is the chosen hidden dimension.

$e_u \in \mathbb{R}^{n \times d}$ is then passed through the first transformer block which consists of a scaled dot-product attention (SA) layer followed by a pointwise feedfoward (FFN) layer as characteristic of transformers. The following equations taken from [14] depict this,

$$S = \mathrm{SA}(E_u) = \mathrm{softmax}\left(\frac{(E_u W_q)(E_u W_k)^T}{\sqrt{d}}\right) E_u W_v$$

where W_q, W_k, $W_v \in \mathbb{R}^{d \times d}$ are linear projection matrices. Followed by,

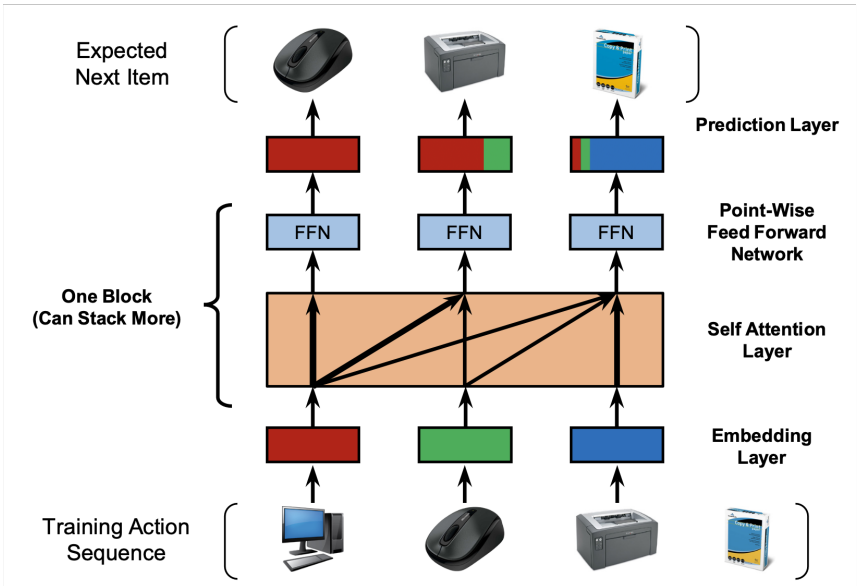


Figure 2.3: SASRec Architecture and Next Item Training. Taken from [14].

$$F_i = \text{FFN}(s_i) = \text{ReLU}(s_i W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$

Hence, a sequence of embeddings $(f_1, ..., f_n)$ is retrieved. This sequence is passed through another transformer block and so on until after k blocks the learned embedding sequence $(f_1^{(k)}, f_2^{(k)}), ..., f_n^{(k)})$ is produced[1]. These embeddings are the learned representation for a user.

The user representations or learned embeddings are used to make predictions. A relevance score is computed for items. The relevance score of some item $i \in I$ at point j in the sequence is,

$$r_{i,j} = f_j M_i^T \tag{2.1}$$

The relevance score is an absolute measure. It is used to calculate the probabilities which depend on the scores of other items as well. These final probabilities are calculated using a simple softmax function across all items in consideration.

An important feature of the SASRec is causal masking. This prevents any interaction in the sequence from learning on future interactions. That is, the i-th position only has access to the previous or $(j \leq i)$ positions during the self-attention stage [14].

[1]We ignore the (k) in future notation and assume f_i denotes an embedding retrieved after all the blocks.

The SASRec also makes use of dropout layers, residual connections and layer normalisation. Lastly, from (2.1), it can be seen that M is shared. This is not an issue as transformers introduce non-linearities [14].

2.2.2.2 Learning Objective

The SASRec uses the sequential next item learning objective. It trains by predicting the next item at each point of the input sequence (See Fig. 2.3). For this, the input sequence S above is actually created with interactions up till the penultimate interaction. This is because the final item has no proceeding interaction to learn on. Then the loss (Binary Cross Entropy Loss) for each user is defined as follows :-

$$-\sum_{k\in\{m-n-1,m-n,\ldots,m-1\}}\left[\log(\sigma(r_{a_{k+1},j}))+\sum_{q\notin S}\log(1-\sigma(r_{q,j}))\right]$$

where qs are randomly sampled negatives from $I/\{S\}$ and $j = k-m+n+1$ (embeddings are indexed from 1 to n). The original SASRec uses one negative sample q for each positive. Padded positions are ignored in the loss computation.

2.2.3 Pinnerformer and Learning on Windows

The Pinnerformer, like the SASRec, uses stacked transformers to understand item relationships in the input sequence and produce learned user representations. The transformer blocks consist of self-attention layers followed by pointwise feedforward layers like SASRec.

The major dissimilarities present themselves in the manner in which the two models learn [2]. The Pinnerformer is optimised for the window-based prediction task introduced in Section 2.2.1.2. Instead of learning on the next item, or every next item at each point in the user sequence as in SASRec, the Pinnerformer learns on positive samples from a future window.

2.2.3.1 Learning Objective

The Pinnerformer creates input sequences and windows from each interaction sequence. Global time point T is taken such that a window of the chosen time period is created from T to the current time. A user interaction sequence is divided such that interactions

[2]Other details such as formation of item embeddings, activation functions, loss function etc. are different as well. However, for the purpose of this paper we are interested in the contrast between the learning mechanisms, other developments are considered orthogonal.

that occur before T are assigned to the input sequence[3] and interactions occurring after T are assigned to the window. The Pinnerformer considers windows in days and experiments primarily with 14-day windows.

The input sequence is passed through the transformer blocks to produce learned embeddings. The embeddings are trained on positive samples from the window. The learned embeddings train on the positive samples in two ways :-

- **All Action:** For each user, predictions are only made on the final embedding through which the loss is propagated backwards.
- **Dense-All Action:** For each positive sample, a random embedding is selected from the sequence of learned embeddings make predictions. Loss is propagated through multiple embeddings.

The different ways of learning on the embeddings produced define the two window-based learning objectives introduced in the paper. Both are discussed in detail in Section 3.4 alongside more novel objectives introduced in this work.

2.2.4 Transfer Learning

Transfer learning (TL) involves training a network on a source domain, a source task or both and using the knowledge acquired to improve learning on a target domain and task. Domain alludes to the feature space and the marginal distribution over the feature space. The task is described by the label set and the objective prediction function [23]. The general idea is that the domains or tasks between the source and target are related, hence, *pretraining* on the source results in better learning when eventually *finetuning* on the target domain/task. In practice, training on weights already optimised for the source give better results than training from scratch with random initialisation for the target [4, 36]. Naturally, this depends on the relationship between the source and target.

2.2.5 Fixed Period Time2Vec

Time2Vec (T2V) [15] proposes the following method to capture periodic information -

$$\mathrm{F}\left(\frac{2\pi}{p_i}T+\varphi\right)$$

[3]There is an intermediary truncating or padding step to ensure input sequences are of the same length.

where T is some timestamp, F is a periodic activation function, and p_i is the time period and φ is the phase shift for the activation function. Hence, $\frac{2\pi}{p_i}$ is frequency of the activation function which is usually a sinusoid. Or, Ts divisible by p_i will have the same value for a given p_i. In this manner, a time feature vector can be created by learning on multiple periods to capture periodic information quite comprehensively. The idea motivated by the Fourier Transform [13], which allows us to break complex time-series into sums of their composite sinusoidal parts.

The original T2V learns the relevant periods p_i; however, the Pinnerformer [24] proposes the use of fixed time periods. Periods are chosen such that they have particular relevance to human concepts of time; for example, 1 minute, 15 minutes, 1 day 1 week, etc. Furthermore, the Pinnerformer uses Sine and Cosine periodic activations, producing, for each period, two embeddings.

Chapter 3

Methodology

3.1 Dataset

The 20M MovieLens (ML-20M) dataset [7] is used [1]. It consists of 20 million movie ratings made by different users. The movies are treated as the "items" that users interact with. The act of rating a movie is the action. We do not consider the value of the rating as in [14]. Hence, future movies to be rated are predicted based on sequences of previously rated movies by users.

The dataset is a larger version of the common benchmarking ML-1M dataset, which has 1 million ratings, for RSs. The 20M dataset was chosen because offline analysis with temporal windows requires interactions for a user to span over a longer temporal range. We choose a window size of 14 days to stay consistent with [24]. Only 1650 users out of 6040 total users in the ML-1M have sequences that span for more than 14 days. This is insufficient for statistically credible results.

In the 20M dataset, 38603 users have interactions spanning for more than 14 days. This subset of users is used for experimentation. The number of unique items (or movies) interacted with is 26402. Table 3.1 presents before and within the window statistics where windows are defined as the last 14 days of interactions for each user.

	Mean	Median	Lower Quartile	Higher Quartile
Before Window	272	156	65	338
Within Window	25	7	2	23

Table 3.1: Statistics for User Interactions Before and Within the Future Window

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

3.2 Splitting Strategy and Window Creation

The splitting strategy encapsulates the way in which the interactions sequences are split to define the training, validation and test sets or windows. We adopt 2 splitting strategies - (a) user-first and (b) time-only. These strategies might not be best practice but are used because of data constraints. This is explained in detail in this section.

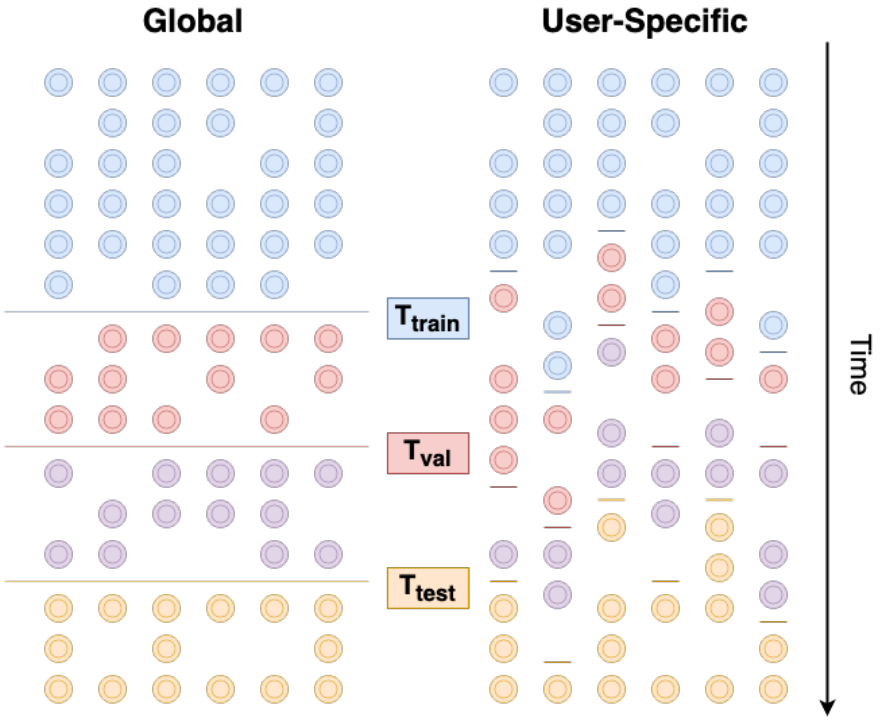


Figure 3.1: Global vs User-Specific Temporal Splitting. On the left, splits are made at global timestamps. On the right, splits are at different timestamps for each user.

3.2.1 User-First

The user-first strategy splits by users first. 70% of users are taken for training and 15% each for validation and test. Following this, input sequences and windows are created for all the users in each set in a similar manner to the Pinnerformer (See Section 2.2.3.1). However, instead of a global time point, user-specific time points are taken (See Fig. 3.1). For each user, a window of the chosen time period is created by counting backwards from the time of the last interaction for that user to time T_u. That is, the input sequence $S = (a_{k-n}, ..., a_k)$ such that $t_k < T_u$ and the window $W = \{a_k | t_k \geq T\}$ where t_k is the timestamp of interaction a_k. We consider 14-day windows.

This strategy does not perfectly mimic the online use-case. First, in reality, we train and predict on all users. Here, evaluation is done on a different subset of users which could make the task more challenging for the model and bias the results downwards. Second, in a deployment scenario there is some absolute timestamp till which updated user interactions are available because we cannot look into the future. All users are blind to information after this point. In our case, windows are user-specific. Some users in the training set could have training windows further in the future relative to windows of some users in the evaluation sets (Fig. 3.1). That is, there is not complete blinding to future information for all users. This can bias results upwards. However, due to dataset constraints, we are forced to use the above strategy to retain a good population of users.

Ideally, experiments should use the global splitting strategy (See Fig.3.1) to mimic a deployment scenario offline. However, this strategy is infeasible on the 20M dataset because users tend to have non-overlapping periods of activity: in any chosen 14-day window, only a few users show activity. From Fig. 3.2 it can be seen that the maximum number of users interacting in any 14-day window is only 1676.

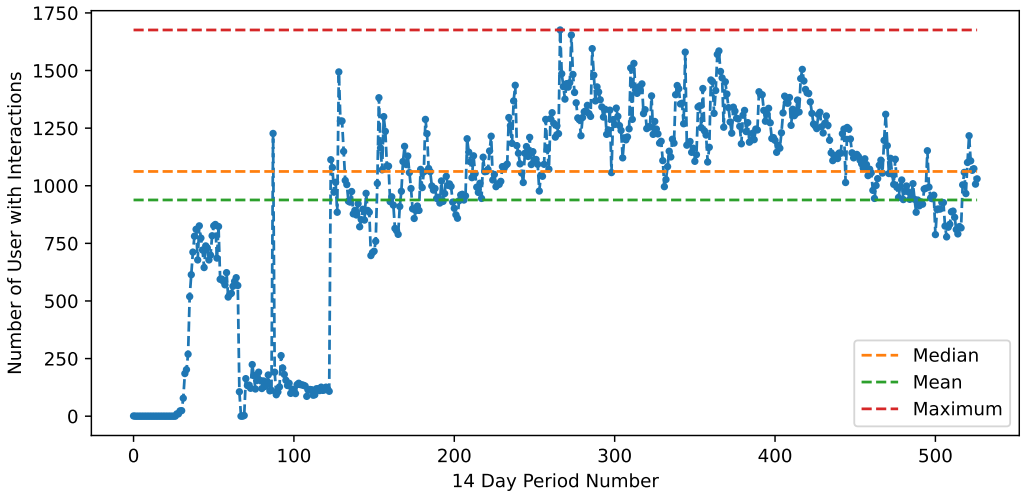


Figure 3.2: The figure divides the total time period on which there is data into 14-day windows and depicts the number of users active in each of these windows.

3.2.2 Time-Only

The time-only strategy directly employs user-specific time splits without splitting sets based on users first. However, the validation window is not created. Only 3549 users in the 20M dataset have interactions in all three train, validation and test 14-day windows

with user specific splits. On the other hand, 9719 users have interactions in both the final two 14-day windows. Hence, to reduce data loss only training and test sets are created and the best epoch is directly selected on the test set.

Even then, this strategy results in the loss of 28884 users compared to the user-first strategy. It is used because it is necessary for experimental robustness when comparing once vs daily inference (See Section 3.7). The daily inference should theoretically see gains in performance because it is privy to the latest interactions by a user made on the previous day. If a user split was done before, then this latest information would not necessarily extrapolate well to users outside the training set. Hence, a time-only split is employed to ensure results are not biased because of this reason.

3.3 Evaluation Metrics

Two Top-N metrics are used to evaluate the performance of the models[2]. We also look at the diversity in predictions made as secondary measure. Finally, the temporal performance of the models is discussed. Evaluation is performed on future windows.

3.3.1 Top-N Performance

3.3.1.1 Recall

The Recall@10 takes the average of the number of times positive samples are ranked in the top 10 items by our model. This is averaged over the number of positive samples for each user and the number of users. The equation taken from [24] below illustrates this,

$$Recall@10(u) = \frac{1}{|W_u|} \sum_{p \in W_u} \mathbf{1}\{|\{n \in N | r_p \geq r_n\}| < 10\} \tag{3.1}$$

$$Recall@10 = \frac{1}{|U|} \sum_{u \in U} Recall@10(u)$$

where U is the user set, W_u is the window for each user and r is the relative rank of the items. N is a set of 500 negative items randomly sampled for each positive sample.

3.3.1.2 Normalised Discounted Cumulative Gain (NDCG)

The NDCG@10 is similar to the Recall but is position aware [14]. It not only considers whether or not a positive sample ranks in the Top 10 but also the rank. Mathematically, the portion inside the summation in (3.1) is replaced with $\frac{1}{\log_2(r_p+2)}$

[2]Performance and Top-N performance are used interchangeably throughout the paper.

3.3.2 Diversity

We look at the global variation in predictions made by the model using P90 metric. It calculates the fraction of items that are in the 90th percentile of being ranked in the top 10 items for users [24]. A predefined set of items is ranked across models for consistency. Lower values indicate that the model is recommending similar items for all users while higher values suggest it is making different predictions for different users.

3.3.3 Temporal Performance

Temporal performance is judged in terms of the average time per epoch over all epochs and the total wall clock time to the best epoch. The total wall clock time is calculated by taking the average time per epoch $\times$ the best epoch number.

3.4 Window-Based Learning Models

This section describes the various window-based learning models. These models are different from the original SASRec in that they optimise using window-based learning objectives. The SASRec is modified for these new objectives. The changes are characterised by how the input sequence, the positive samples and the loss is defined. These aspects resemble the Pinnerformer (2.2.3).

The input sequence S is passed through the SASRec (See Section 2.2.2.1) to retrieve the learned embeddings $(f_1, f_2, , ..., f_n)$. Positive samples are randomly taken from the window W, created as explained in Section 3.2.1, with a fixed upper limit on the number of samples per user. Negative samples are randomly selected for each positive sample avoiding interactions in a users interaction sequence. We continue to use relevance scores and the BCE loss function as in SASRec [14].

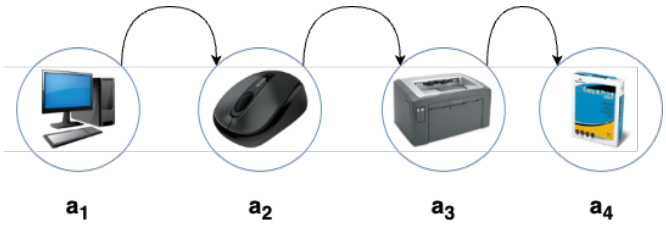


Figure 3.3: Simplified Diagram of the Sequential Next Item Training for comparison with new objectives. Item images taken from [14].

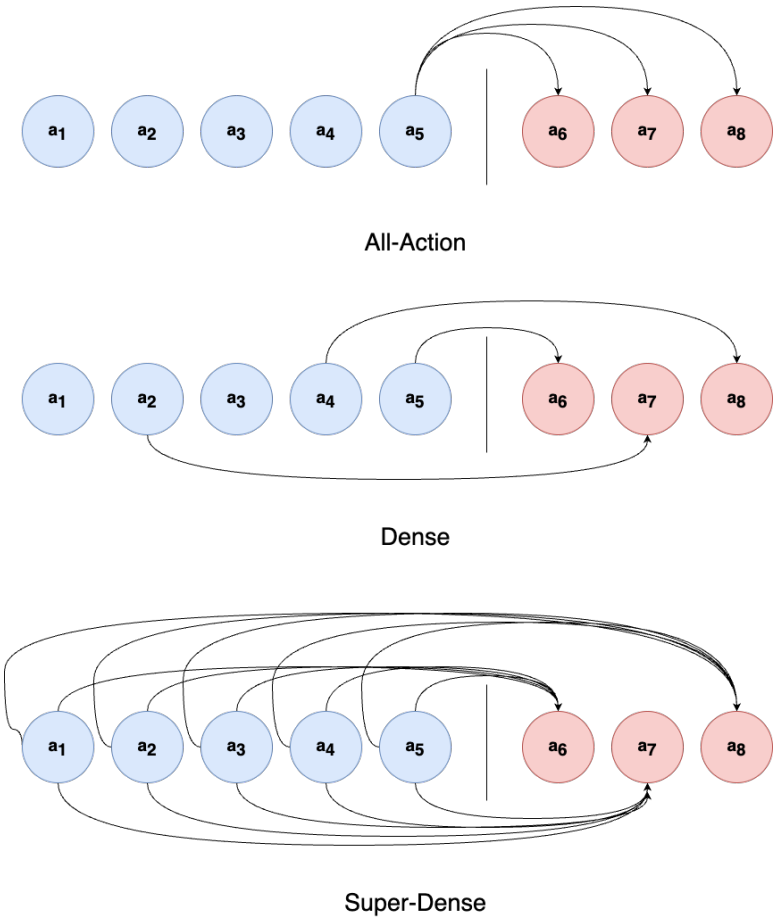


Figure 3.4: Simple Representations of All Action, Dense All Action and Super-Dense Learning Objectives. The vertical line separates the input sequence from the training window. Arrows indicate which positions in the input sequence, or which learned embeddings, train on which positive samples from the window.

3.4.1 All Action

The all action model trains only on the final learned embedding (See Fig. 3.4). For each positive the relevance score is calculated,

$$r_{p,n} = f_n M_p^T$$

where f_n is the final learned embedding, p is the positive sample from W and M is the item embedding matrix. Then, the loss is computed as follows,

$$-\left[\log(\sigma(r_{p,n}) + \sum_{q \notin S \cup W} \log(1 - \sigma(r_{q,n}))\right]$$

where qs are the sampled negatives. During inference, predictions are made on the

basis of the final learned embedding. The name "all action" comes from the fact that we are learning on *all* positive samples in the window rather than just the next item[3].

3.4.2 Dense All Action

The dense model[4] trains a different learned embedding for every positive sample. The embedding is randomly selected. Padded positions are ignored. The loss,

$$-\left[\log(\sigma(r_{p,j(p)}) + \sum_{q \notin S \cup W} \log(1 - \sigma(r_{q,j(p)}))\right]$$

where $j(p)$ is chosen randomly for each p. Predictions are made only on the final embedding during inference. It is "dense" as more learned embeddings are used.

3.4.3 Super-Dense All Action

The super-dense learner trains *all* the learned embeddings on every positive sample. This is why it is called "*super*-dense" It computes

$$-\left[\sum_{j \in \{1,2,\ldots,n\}} \left[\log(\sigma(r_{p,j})) + \sum_{q \notin S \cup W} \log(1 - \sigma(r_{q,j}))\right]\right]$$

Embeddings that correspond to padded locations do not contribute to the loss. Notice, each embedding makes an individual contribution to the loss. Inference is still on the final embedding.

3.4.4 Target-Dense

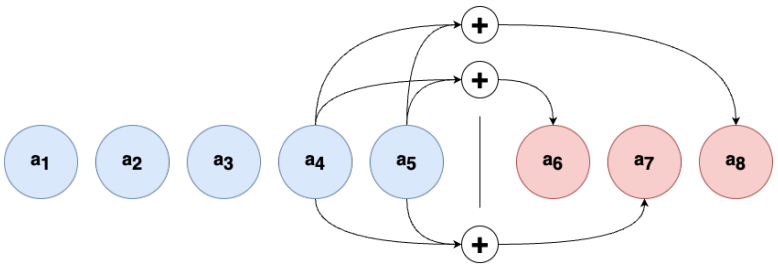


Figure 3.5: Target-Dense Learning Objective

[3]Even though there is an upper limit on the number of positive samples taken for each user, the positives are sampled randomly in every epoch. This bootstrapping allows us to train on the whole window on average.

[4]"All-action" is dropped in places to avoid redundant and wordy nomenclature.

Inspired by similar ideas to super-dense model, the target-dense learner uses the final b learned embeddings for training. However, instead of learning individually as in super-dense, a weighted sum of the relevance scores produced with each selected embedding is taken to produce a single score for any sample. The loss is computed on the basis of this weighted sum. That is, embeddings make a combined contribution to the loss.

$$-\left[\log\left(\sigma\left(\sum_{j\in\{n-b,\ldots,n\}} w_j r_{p,j}\right)\right)+\sum_{q\notin S\cup W}\log\left(1-\sigma\left(\sum_{j\in\{n-b,\ldots,n\}} w_j r_{q,j}\right)\right)\right]$$

where we take the final b learned embeddings to train on and w_j are learnable scalar weights. Furthermore, what distinguishes target-dense from other objectives is that the selected embeddings are used during inference as well. The target-dense can learn to focus or "*target*" the most relevant embeddings for prediction.

3.5 Combined Learning Models

We suspect that the sequential learning ideas and task specific learning ideas offered by the next item learning objective and window-based learning objectives respectively are related. This is explored by combining the two ideas in interesting ways. Two methods are used - a combined objective that incorporates sequential ideas into window-based learning and transfer learning. Finally, a hybrid model is introduced to understand if there is an optimal balance between the two learning ideas that gives the best performance on the window prediction task. These 3 methods define the combined learning models used in our research.

3.5.1 Window Next Item

The window next item learner combines the sequential next item learning objective with the window-based objective. As in the SASRec, every embedding in the sequence learns from the very next item in the sequence; however, the final embedding trains on items from the future window. The loss is defined as,

$$-\left(\sum_{k\in(1,2,\ldots,n-1)}\left[\log(\sigma(r_{a_k+1,k}))+\log(1-\sigma(r_{q_k,k}))\right]\right)-(\log(\sigma(r_{p,n}))+\log(1-\sigma(r_{q_n,n})))$$

where $S=(a_1,a_2,...,a_n)$ is the input sequence and q_is are randomly chosen negatives such that each $q_i \notin S\cup W$. As we can see the summation resembles the loss in

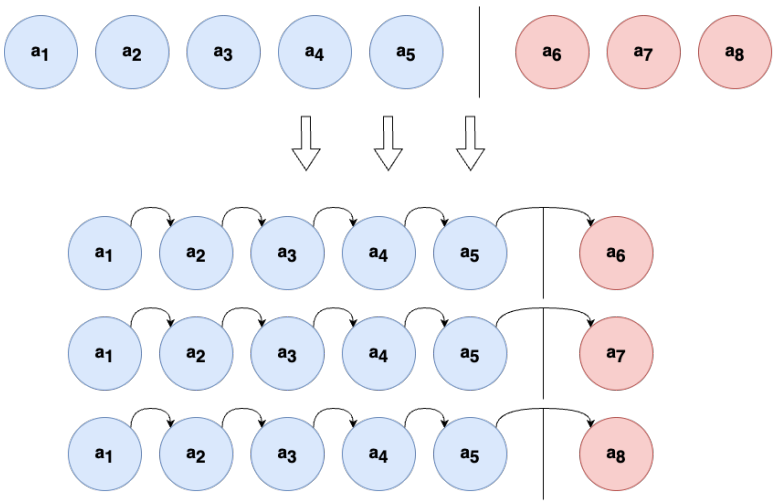


Figure 3.6: Window Next Item Learning Objective

Section 2.2.2.2. It is the second half of the equations that defines the window learning component.

3.5.2 Transfer Learning

To further unravel the relationship between the tasks, we pretrain on the sequential next item objective and fintune on the window-based objectives. The next item objectives correspond to the source task, next item prediction, and window-based objectives to target task of window-based prediction. The source and target domains remain consistent [5]. If time features are included, they are included for the source as well as the target models. During the finetuning stage, a smaller learning rate (= 0.0001) is chosen. No weights are frozen, nor additional layers added.

The idea is that if the learning objectives are related, the knowledge transfer will enhance the performance of the window-based objectives.

3.5.3 Hybrid Model

The window next item and transfer learning models illustrate that there is a relationship between the learning ideas (seen later). Then using sequential learning in combination with window-based learning improves performance. However, we aim to understand if there is an optimal balance to be struck between the two learning approaches to maximise performance.

[5]Pretraining is also done on the entire 20M corpus as opposed to the subset of users with more than 14 days of interactions. In this case, the source and target domains differ (See Section 4.3).

In the window next item learning model, the same input sequence is repeatedly trained on for every positive sample (See Fig.3.6). This might cause the window next item to *overemphasise* sequential ideas. Some more performance could be found if general sequential learning is balanced with task specific learning more equally.

The hybrid model combines window next item learning with all action learning. The hybridisation is achieved using a cyclical approach: the objectives are cyclically switched step-wise or batch-wise during training according to a predefined (cycling) ratio. We experiment with cycling 1 step to 1 step (1:1), 1 step to 3 steps (1:3), and 1 step to 9 steps (1:9) of window next item and all action respectively.

The all action objective has no sequential learning component. Hence, it helps counteract over-fixation on sequential ideas. We experiment with the different cycling ratios to find the optimal balance. Moreover, since we are not constantly learning on the whole sequence, the hybrid objective can help reduce the time complexity of the window next item. This is also why we adopt a cyclical approach rather than simply using a hybrid loss function that uses weights to combine the objectives [6].

3.6 Adding Time Information

To understand if time information can improve the performance of the various models, periodic time features are added using the fixed period T2V described in Section 2.2.5. A vector of periodic time information is concatenated to the item embeddings. We use 7 periods[7] giving a total of 14 additional features taking sine and cosine activations. The new dimension of input embeddings is now $d + 14$, or $E_u \in \mathbb{R}^{n\times(d+14)}$. However, for this study predictions within the time windows are time-agnostic. Hence, the additional dimensions need to be accounted for so that the learned embeddings used to compute relevance scores have a latent dimension d. To do so, a Multilayer Perceptron (MLP) is added on top of the SASRec that bumps up the dimensions of the final embeddings $\times 4$ and then brings it down to the required dimension. This architecture with the extra MLP layer referred to as T2VSASRec in the rest of the project.

[6] Preliminary experiments were conducted with this approach. We found similar results for similar ratios, but learning was much slower.

[7] 1 minute, 10 minutes, 1 hour, 12 hours, 24 hours, a week and a month

3.7 Once vs Daily Inference

For **RQ4**, we want understand whether the new window-based and combined models can enable longer gaps between model updates in deployment scenarios. In our case, a 14-day (the chosen window size) and a daily update cycle is compared. To simulate and experiment with these ideas the following approaches are adopted :-

1. **Once:** Training is done once before the 14-day window. Performance is evaluated over the window. It mimics a real scenario in which models are updated in cycles equal to the window lengths, eg. every 14 days.

2. **Daily:** Training is done daily to make predictions for the following day in the 14-day window. We train till day K-1 to predict for day K where $K \in (T, T+14)$ [24]. This mimics daily inference, which is widely adopted in RSs, with offline data.

Window-based and combined learners are trained as in (1) are compared with the next item learning model trained as in (2).

3.8 Experimental Details

Models are implemented with Pytorch within the Pytorch Lightning framework. Models are run for 100 epochs and the best epoch is chosen based on the highest NDCG score, calculated every 5 epochs, on the validation set. The models use 2 blocks of transformers with a single head. The maximum sequence length (n) is 50 and the latent dimension (d) is 50. A dropout rate of 0.2 is chosen. For window-based and combined learners, 16 is the maximum number of positive samples allowed per user and 50 negative samples are taken. For Target-Dense learning, the number of learned embeddings (b) trained on is 25. The models are optimised using the Adam Optimiser [16] with a learning rate of 0.001. For TL, a lower learning rate ($= 0.0001$) is adopted during the finetuning stage.

Chapter 4

Results and Discussion

4.1 Window-Based Learning Objectives

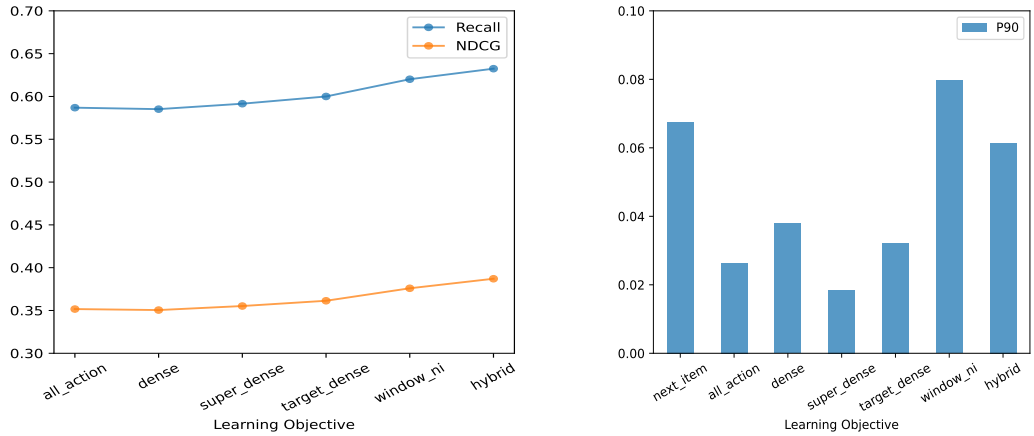
The first set of experiments is aimed at evaluating the performance of the various models using window-based learning objectives on the window prediction task. The 20M dataset with the user-first splitting strategy is used. Table 4.1 below summarises the results.

Learning Objective	Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch
Next Item	0.539	0.310	0.068	21	10
All Action	0.587	0.352	0.026	66	5
Dense	0.585	0.350	0.038	69	15
Super-Dense	0.592	0.355	0.019	116	5
Target-Dense	0.600	0.361	0.032	66	5

Table 4.1: Results for Different Window-Based Learning Objectives on the Window Prediction Task. Best epoch is decided on the validation set and validation is done every 5 epochs.

4.1.1 Top-N Performance

Models that use window-based objectives outperform the next item objective learner on the window prediction task. The experiments confirm our hypothesis for ***RQ1****.* The next item objective gives a R@10 of 0.539 and a NDCG@10 of 0.310. The all action model, the most intuitive and direct window-based learner, improves on this by 9% and 13.6%



(a) Performance (b) Diversity

Figure 4.1: Top-N Performance and Diversity of the Different Learning Objectives

(See Fig. 4.2). This is a significant gain in performance. To illustrate, let us consider recall. Taking a hypothetical dataset of 1000 users with 10 positive interactions each in the future 14 days, all action learner identifies 480 more positive interactions in the top 10 as compared to the next item objective. With 10000 users, this number goes to 4800 interactions and considering 100 positive interactions per user, it already touches 48000 interactions [1]. In certain deployment scenarios, we could be dealing with millions of users or hundreds of future interactions and hence, the gain is significant.

Further, the relative gain is more pronounced on the NDCG which considers the rank of the predictions as well. This shows window-based learning identifies ground truths with more conviction. This is consistent in all the window-based objectives.

The dense model also improves on the next item model. However, the dense learner has similar if not slightly lower Top-N performance than the all action. This disagrees with the results presented in [24] that show that the dense models outperforms the all action model. In the paper, this is attributed to the loss being propagated through different embeddings instead of just the final embedding. This helps mitigate the averaging effect: the gradients only get averaged after passing through the final transformer and not before as with the all action model [24].

It is difficult to pinpoint where the difference in results stems from because of other differences in implementation between our models and those in [24]. However, it seems that all action and dense all action learning objectives exhibit fundamental differences

[1] Here, the positive samples are only evaluated against negatives. In reality, the positive items will also compete with each other during inference. Nevertheless, the metric gives a good indication of the realtime performance.

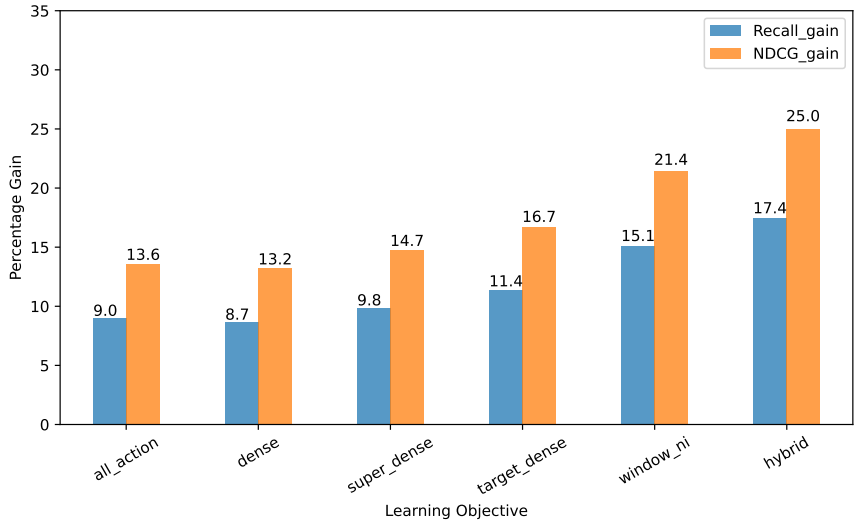


Figure 4.2: Percentage Gain in Top-N Performance of the Different Window-Based Models Relative to the Next Item Model

in learning behaviour that go beyond gradient propagation and averaging effects. This is made apparent by the manner in which the two models pay attention. Fig. 4.3 shows that all action focuses attention on the most recent positions while dense all action tends to acknowledge positions further in the past.

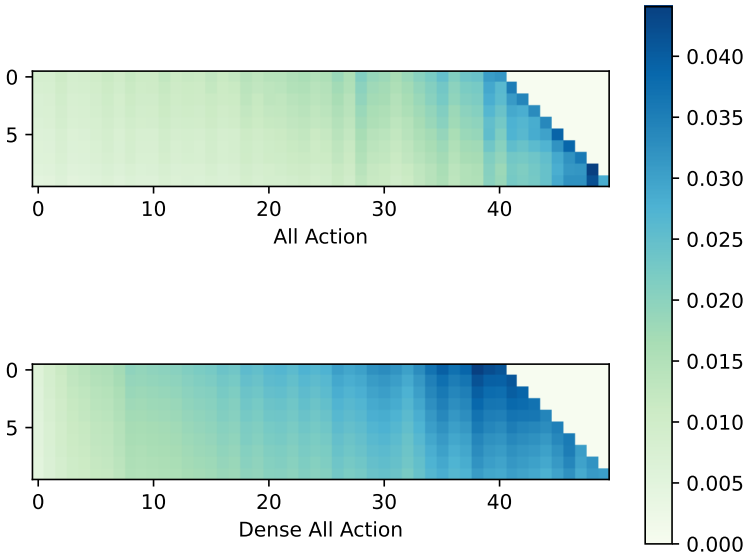


Figure 4.3: Attention Weight Map in the 2nd Transformer Block for the last 10 Positions

Super-dense and target-dense learners also improve on the next item learner. Super-dense shows a 9.8% and 14.7% increase in R@10 and NDCG@10 respectively. The target-dense shows an improvement of 11.4% and 16.7%. Moreover, these novel ideas

slightly edge all action and dense all action learning methods (Table 4.1).

The superior learning of the super-dense model could be attributed to the diversity in training the objective induces. Each learned embedding has access to only a truncated portion, up to the position of the learned embedding itself, of the entire input sequence because of causal masking [14]. When training on every learned embedding, the model is learning to predict with the whole sequence as well as with different truncated sequences. This reinforced learning on the truncated sequences makes the model more robust to variations. For example, let's take a user that engages in a similar sequence of interactions as those the models is already trained on sometime in the future. However, let's say they do some out-of-the-ordinary interactions towards the end. In this scenario, the super-dense model will stick to predictions it would have made if not for the anomalous transactions. This is because the reinforced learning makes the super-dense more confident about predictions based on interactions preceding the anomalous interactions. This is an interesting direction for future work. It could be interesting to understand if learning on subsequences can improve model performance.

The target-dense model, on the other hand, learns and infers based on multiple embeddings. It can decide on how much emphasis to put on the different embeddings and focus on the embeddings that have the more predictive power. Looking at the weights (not presented) that define the contribution of the embeddings the models puts more weight on more recent embeddings.

4.1.2 Diversity

Window-based learners see a fall in global diversity of predictions compared to the next item learner (See Fig. 4.1b). This is a secondary consideration.

Dense action gives more diverse predictions than all action. This coincides with results in [24]. The super-dense model shows the lowest diversity in predictions with a P90 score of just 0.19. The resistance to variation also translates to rigidity in predictions. The model tends to stick to similar predictions because of the reinforced learning discussed in the previous section.

The target-dense learning model, however, is the second most diverse window-based learner. This shows that the target-dense is distinctly different to super-dense even though they share some similar training ideas.

4.1.3 Temporal Performance

The gains in performance come at the cost of temporal performance. The all action, dense all action, target-dense all action models take nearly thrice as much time per epoch as the next item learner. Even though training is no longer done on every item in the sequence as in the next item learner, it is repeated multiple times for the same sequence paired with different positive samples from the training window. This accounts for the increase in time. To illustrate, for 10 users with 16 positive items in the window each, the next item learner trains on each user once, i.e., 10 times in a epoch. The window-based learners, on the other hand, train on each positive item separately; that is, an epoch consists of 10×16 computations.

The super-dense model takes nearly 6 times more time per epoch on average. This is because *every* embedding is learning on each positive item.

Learning Objective	Next Item	All Action	Dense	Super-Dense	Target-Dense
Wall-Clock Time (s)	208	331	1032	578	329

Table 4.2: Summary of Total Wall Clock Times of the Different Learning Objectives

In terms of total wall clock time taken till the best performing epoch, the all action and target-dense models are relatively efficient. They take approximately 120 seconds longer than the next item learner (See Table 4.2). This is because of their quicker convergence behaviour - 5 epochs compared to 10 epochs on the next item learner - partially accounts for the higher per epoch times. For a simple perspective, this equates to about 0.11 percentage point gain in NDCG@10 for every added second of training time for the all action learner.

The dense all action shows slower convergence. The super-dense converges in 5 epochs but the extremely high time/epoch still results in a large total wall clock time. Fig. 4.4 shows the convergence behaviour over a 100 epochs with evaluation run every 5 epochs. Because of the early convergence of a majority of the models, Fig. 4.5 zooms into the first 25 epochs and evaluates on every epoch. The smaller range allows us to observe in terms of wall clock time as well[2].

[2]Window next item and hybrid models are presented in the figures in this section itself to emphasise the difference in performance between window-based and combined learners.

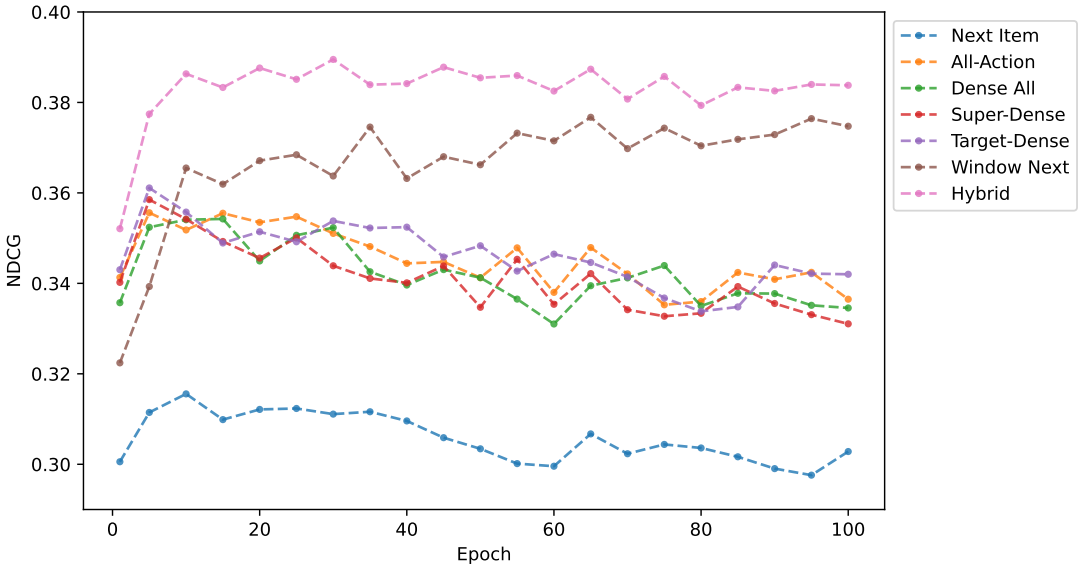


Figure 4.4: NDCG Over 100 Epochs: The figure depicts the convergence behaviour of the various models.

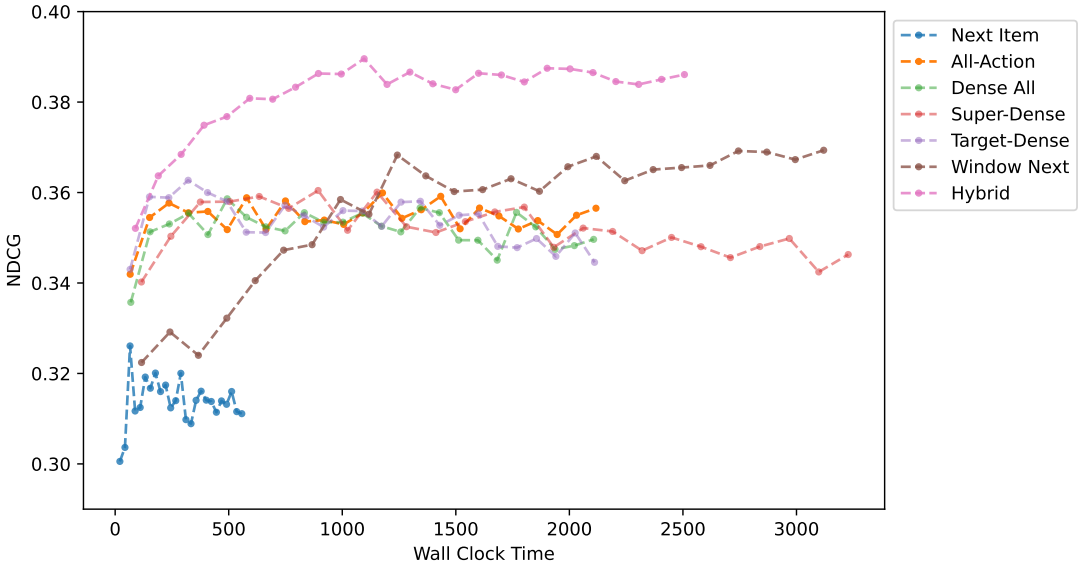


Figure 4.5: NDCG vs Wall Clock Time Over 25 Epochs: Convergence behaviour in terms of time.

4.2 Window Next Item Learning Objective

The first combined model, window next item learner, improves on the next item learner by 15.1% *on R@10 and* 21.4% *on NDCG@10. It outperforms all the window-based models.*

The window next item learner trains the final embedding on interactions in a future

Learning Objective	Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch
Window Next Item	0.620	0.376	0.080	116	65

Table 4.3: Results for the Window Next Item Objective

window. This is task-specific learning. At the same time, it also trains every other embedding to predict the next item. This is related to the task but not specific to the task. It allows the model to pick up more general sequential ideas. *The results show that learning sequential ideas in tandem with task-specific ideas is beneficial. This suggests the two learning ideas are related.*

The window next item learning objective also sees an increase in diversity compared to the next item learner (See Fig. 4.1b). This makes window next item even more impressive. It gives a P90 score of 0.08 which is a 17.8% improvement on the next item learner.

However, the window next item model takes nearly 6 times more time per epoch on average. The combined training ideas mean the window next item learns on every point on the sequence and also sees the multiplicative effect of window learning.

Further, it takes 65 epochs or 7550 seconds to converge (See Table 4.7). This is expensive, especially in realtime when operating with thousands or even millions of users and items. However, as the Pinnerformer [24] shows, this increase in time training window-based models can be offset by the gain in performance on the window prediction task. This is because it makes training in window length cycles viable. We investigate this further in Section 4.6.

4.3 Transfer Learning

Learning	Without Transfer Learning			With Transfer Learning		
Objective	Recall@10	NDCG@10	P90	Recall@10	NDCG@10	P90
All Action	0.587	0.352	0.026	0.616	0.374	0.018
Dense All Action	0.585	0.350	0.038	0.612	0.371	0.022
Super-Dense	0.592	0.355	0.019	0.611	0.371	0.015
Target-Dense	0.600	0.361	0.032	0.611	0.371	0.023

Table 4.4: Side by Side Performance of Models With and Without Transfer Learning

The window next item model outperforms all other learning objectives on the window prediction task. This suggests, as hypothesised for ***RQ2****, there is some kind of relationship between the next item learning and window-based learning. Here, we directly exploit this idea by using transfer learning (See 3.5.2).* Models are pretrained on the next item learning objective and finetuned on the window-based objectives - all action, dense, super-dense and target-dense.

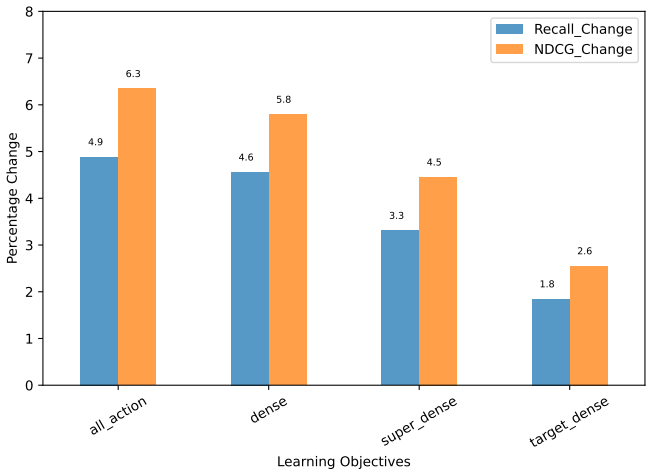


Figure 4.6: Percentage Gain in Top-N Performance with Transfer Learning

We see that transfer learning improves performance but reduces diversity in predictions for all four models. Table 4.4 above summarises the performances of the transfer learning models. Fig. 4.6 shows the percentage improvement in performance on the models with and without transfer learning. All-action sees the biggest jump in Top-N performance and gives the best performance among the transfer learning models.

Table 4.5 presents the best epochs and the total wall clock time (including the time taken to pretrain). We can see the times are similar to times in Table 4.2 but for the additional time required to pretrain the next item model. Even with the additional pretraining time, transfer learning models are cheaper than window next item and boast performances close to that of the window next item. In fact, all action with TL sees only a difference of -0.004 R@10 to the window next item.

Learning Objective	All Action	Dense	Super-Dense	Target-Dense
Best Epoch	5	15	5	5
Wall-Clock Time (s)	521	1244	766	576

Table 4.5: Summary of Total Wall Clock Time for the Different Models with TL

*The results further substantiate argument for **RQ2**. Sequential next item and window-based learning are related objectives; window-based learning can use knowledge acquired through sequential next item learning to improve its performance on the window prediction task.*

Till now, we only train on the subset of users with more than 14 days of interactions (See 3.1). Recently, Large Language Models (LLMs) like GPT [22] and BERT [2] have shown that pretraining on more general source tasks improves the performance on the target tasks because of the larger corpus of data available for the general tasks compared to the target task [26]. The next item objective does not require more than 14 days of interactions by users for training. In other words, next item learning has access to the entire 20M dataset. Hence, we pretrain on the entire 20M dataset with the next item learner and finetune on the window-based learners on the subset of users. In this case, the source and target domains also differ as the marginal distribution over features is different between the complete 20M dataset and its subset.

We choose to only run the experiment with the all action window objective as the target because of its superior performance earlier. Surprisingly, a 0.604 R@10 and 0.365 NDCG@10 is seen[3]. This is an improvement on the vanilla all action but a drop in performance compared to the case above. It seems that the two learning objectives are linked but *overlearning* on sequential ideas might detract from the window prediction task. This agrees with findings in the next section.

4.4 Hybrid Learning

Based on the experiments in the previous section, the sequential and window-based learning are related. Moreover, using them together enhances quality of predictions. *In this section, we try to see if it could be useful to find an empirical balance in learning sequential ideas and task specific ideas. The hybrid learner cycles between the window next item and all action objectives during training. Window next item has a sequential next item learning component whereas the action objective only learns on window samples. Hence, interspersing window next item learning with all action learning can reduce time spent training on sequential ideas.*

We experiment with different cycling ratios to find the best performance. From Fig. 4.7 it is clear to see that there is an optimum cycling ratio at which the combination of the sequential ideas gives the strongest performance. Alternating the two objectives

[3]See Appendix A.1 for diversity and time performance.

Cycling Ratio	Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch
(1:1)	0.632	0.387	0.061	89	30
(1:3)	0.626	0.382	0.053	77	10
(1:9)	0.611	0.370	0.069	74	20

Table 4.6: Results for the Hybrid Objective with Different Cycling Ratios

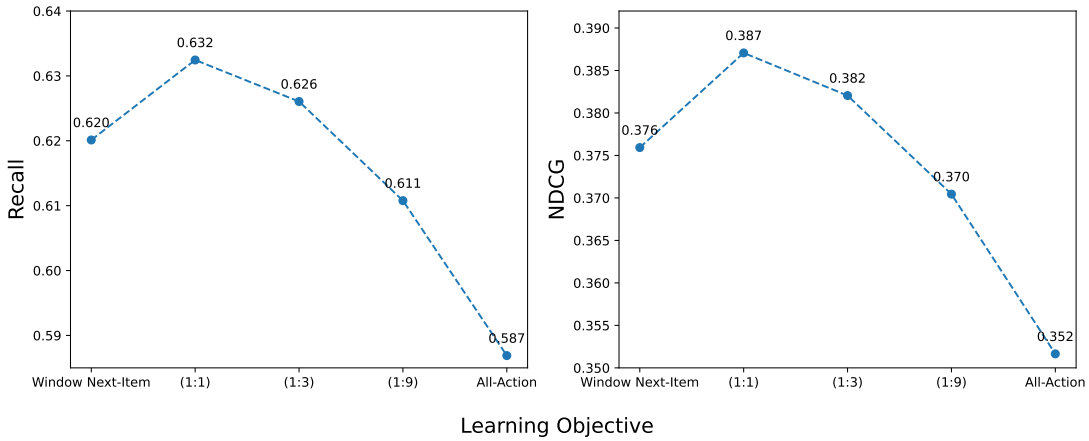


Figure 4.7: Graphical Depiction of Performance with Different Cycling Ratios on the Hybrid Learning Objective. Window Next Item and All Action shown as edge cases for comparison.

or 1:1 ratio, gives the best performance. It improves on the vanilla window next item. Moreover, as the ratio is skewed more and more to favour all action learning, the performance drops with vanilla all action learning objective showing the lowest performance.

The 1:1 hybrid model improves on the vanilla window next item by 2% on R@10 and 3% on NDCG@10. This hybrid model gives the best performance among all time-agnostic models experimented with. It improves on the next item learner by 17.4% and 25%. Even though it sees a drop in diversity of prediction compared to next item and window next item, it shows a P90 of 0.61 which is higher than the window-based objectives. Furthermore, it takes 89 seconds per epoch on average. This is 27 seconds less than the window next item learner. It also converges quicker reaching the best performance in 30 epochs. *In terms of wall clock time, it converges 4877 seconds faster than window next item. We should note, this is still considerably slower than the next item learner (see below).*

*To summarise for **RQ2**, combining sequential and window-based learning improves*

Learning Objective	Next Item	All Action	Window Next Item	All Action TL	Hybrid
Wall-Clock Time (s)	208	331	7550	521	2673

Table 4.7: Total Wall Clock Times of Different Models: Putting time taken by combined learning methods in perspective;.

the performance of window-based objectives. The window next item, transfer learning and hybrid models outperform models that have no sequential component - all action, dense, super-dense and target-dense models trained from scratch. There is an optimal balance between the training ideas. Tuning to find this balance can help extract the best performance. However, the jump in Top-N performance comes at the cost of longer training times. This jump in total wall clock time is more pronounced for window next item and hybrid models as compared to transfer learning models which see only the additional time from pretraining. As discussed later this increase in training time can be counteracted by spaced deployment cycles (See Section 4.6).

4.5 Effect of Time Features

The T2V + SASRec architecture is used to incorporate time information. *For the different window-based learning objectives apart from the target-dense, including time information improves the performance but reduces the diversity of predictions. The window next item and hybrid*[4] *models also see an increase in performance.*

Table 4.8 shows the time-agnostic performance side by side with the time-aware performance of the various models. Fig. 4.8 depicts the percentage change on addition of time features for the different models. *In general, adding time features seems to improve performance as hypothesised in **RQ3**.*

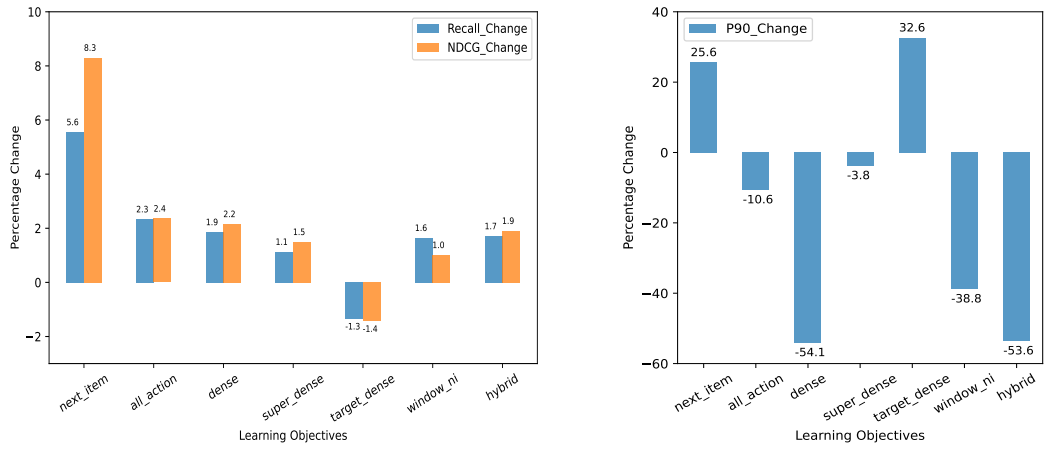
In fact, as can be seen, the next item learner sees the largest benefit of including time features. The next item learner increases by 5.6% on R@10 and 8.3% on NDCG@10. The P90 jumps up to 0.085, a 25.6% increase. It now gives the most diverse predictions among all the models.

However, even with the sharp increase for the next item, the window-based and combined objectives continue to show a better Top-N performance. *The hybrid model remains the model with the best Top-N performance. It has a* 13% *higher R@10 and*

[4]The 1:1 ratio is used because it outperforms the other two configurations even with the addition of time features (See Appendix A.2).

Learning Objective	Time-Agnostic			Time-Aware			Change in
	Recall@10	NDCG@10	P90	Recall@10	NDCG@10	P90	Time/Epoch (s)
Next Item	0.538	0.310	0.068	0.569	0.335	0.085	1
All Action	0.587	0.352	0.026	0.601	0.360	0.024	19
Dense All Action	0.585	0.350	0.038	0.596	0.358	0.017	16
Super-Dense	0.592	0.355	0.019	0.598	0.361	0.018	14
Target-Dense	0.600	0.361	0.032	0.592	0.356	0.043	19
Window Next Item	0.620	0.376	0.080	0.630	0.380	0.049	9
Hybrid	0.632	0.387	0.061	0.643	0.394	0.029	12

Table 4.8: Results for Different Learning Objectives with Time Features Added. Time agnostic results for the objectives presented alongside for comparison.



(a) Top-N Performance

(b) Diversity

Figure 4.8: Percentage Change in Top-N Performance and Diversity for the Different Learning Objectives on Adding Time Information

17.6% *higher NDCG@10 compared to the time-aware next item model.*

Even though adding time features benefits performance, the models take longer to learn. The increase in average per epoch time is not large as shown in Table 4.8; however, the models take longer to converge. Fig. 4.9 shows the convergence behaviour over a 100 epochs with validation performed every 5 epochs. Table 4.9 enumerates the best epoch and the total wall clock time taken to train till the best epoch for the different models. We see a marked increase in the time taken for models to reach optimal configurations. Window next item and hybrid models take the longest to converge, taking the whole and 5 epochs from the whole 100 epochs respectively[5].

[5]To make sure models are in fact converging or close to convergence in 100 epochs, both models are re-run for 150 epochs as well. The results can be seen in Appendix A.3 and confirm this is true.

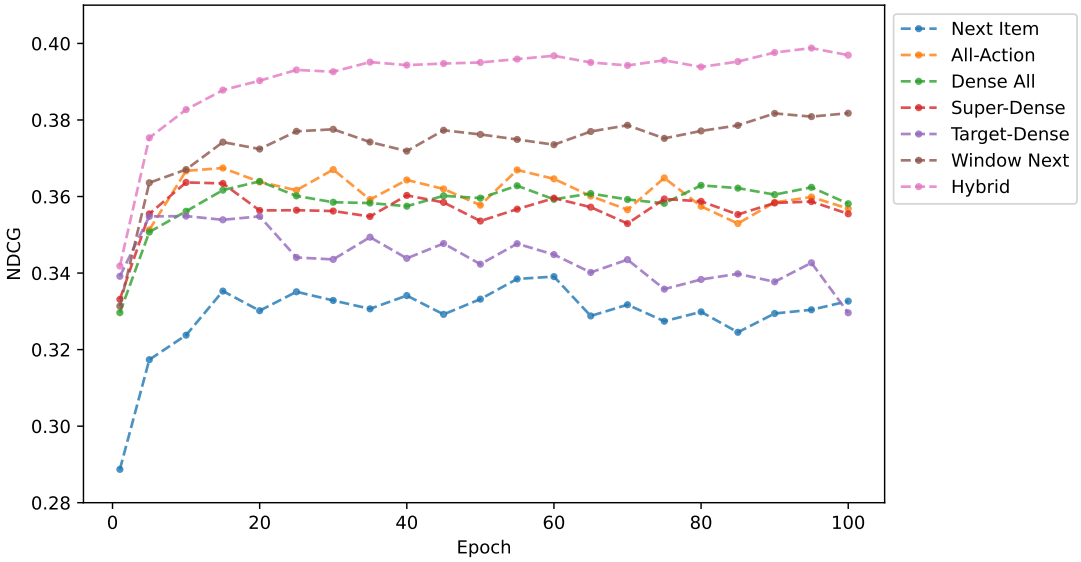


Figure 4.9: NDCG Over 100 Epochs: Convergence behaviour for time-aware models.

Learning Objective	Next Item	All Action	Dense	Super-Dense	Target-Dense	Window Next Item	Hybrid
Best Epoch	60	15	20	10	10	100	95
Wall-Clock Time (s)	1342	1282	1698	1297	852	12515	9569

Table 4.9: Summary of Total Wall Clock Time for Time-Aware Models

Transfer learning improves performance for time-aware models as well. Here, time features are included for the next item learner during pretraining and for window-based learners during finetuning. The change in performance between time-agnostic and time-aware transfer learning is less obvious (See Table 4.10). All-action and target-dense see a drop in performance whereas dense all action and super-dense see a small increase. Contrary to results for models that are trained form scratch, adding time information actually improves diversity of predictions for TL models. However, overall, the time-agnostic TL models achieve similar performance in a fraction of time compared to the time-aware TL models (See Appendix A.4).

*To answer **RQ3**, time features improve model performance but negatively effect diversity and temporal efficiency.* The TL models show slightly different behaviour.

4.6 Once vs Daily Inference

*This section addresses **RQ4**. The time-aware next item learner is trained and evaluated **daily** over a future 14-day window. On the other hand, the time-aware all action and*

Model	Time-Agnostic			Time-Aware		
	Recall@10	**NDCG@10**	**P90**	**Recall@10**	**NDCG@10**	**P90**
All Action TL	0.616	0.374	0.018	0.609	0.369	0.031
Dense TL	0.612	0.371	0.022	0.616	0.375	0.024
Super-Dense TL	0.611	0.371	0.015	0.617	0.376	0.025
Target-Dense TL	0.611	0.371	0.023	0.602	0.363	0.037

Table 4.10: Side by Side Comparison of Transfer Learning Models With and Without Time Information

*1:1 hybrid models are trained **once** before and evaluated over the same window.* The all action is selected because it is the best performing window-based model. The hybrid combined learning model is the best performing model among all the models evaluated in this study. The time-only splitting strategy is used resulting in a set of 9719 users for reasons explained in Section 3.2.2. Table 4.11 below shows the weighted (by the number of interactions on that day) average performance of the daily learner over the window, and of the window-based and combined models on the same window.

Model	Recall@10	NDCG@10	P90
Daily Next Item	0.526	0.294	0.042
Once All Action	0.532	0.308	0.018
Once Hybrid	0.569	0.335	0.036

Table 4.11: Average Performance for Daily Next Item Learner and Performance of Window-Based Learners Over a 14-day Future Window

It can be seen that the window-based objective trained once actually gives a better performance than the next item learner trained daily on the window. The all action shows a 1.3% and 4.6% gain on R@10 and NDCG@10 respectively[6]. *The combined model shows an even more significant increase.* The hybrid model improves by 8.3% on the R@10 and 14.1% on the NDCG@10.

The window-based and combined techniques have lower diversity than the daily next item learner. However, the combined hybrid learner is able to maintain a relatively higher level of diversity, showing only a 13.4% drop on the P90 score of the next item

[6]Performance for the next item trained once before the window is shown in Appendix A.5 for reference.

learner.

These results are impressive because they show window-based and combined models do not need to be constantly updated. This has the potential to save time and resources in deployment scenarios by allowing for less frequent updates of the models. Table 4.12 summarises the total time taken by the next item learner updated daily over the window and contrasts it with the training time of the window-based and combined learner.

We see that window-based learning with a once before window learning strategy is significantly more time efficient than the next item learner with the daily update strategy. The all action learner takes just 6.8% *of the time taken by the daily learner. The combined model takes more time than the window-based model; however, it is still significantly more efficient than the daily next item learner. The hybrid model takes* 19.4% *of the time taken by the daily next item learner.*

Therefore, window-sized update cycles are viable in deployment scenarios using window-based or combined models. The combined learner can provide bigger gains in performance, but is less time-efficient than window-based models.

Model	**Period**	**Time Per Epoch**	**Best Epoch**	**Total Time**
Next Item	Day 1	7	10	62
Next Item	Day 2	7	65	438
Next Item	Day 3	6	15	91
Next Item	Day 4	7	35	223
Next Item	Day 5	7	95	612
Next Item	Day 6	7	100	671
Next Item	Day 6	7	95	628
Next Item	Day 8	7	95	627
Next Item	Day 9	7	50	336
Next Item	Day 10	7	20	128
Next Item	Day 11	7	85	570
Next Item	Day 12	7	95	632
Next Item	Day 13	7	95	618
Next Item	Day 14	16	90	1400
Next Item	14-day Window		Total	7139
All Action	14-day Window	32	15	483
Hybrid	14-day Window	40	35	1383

Table 4.12: Total Wall Clock Time for the Next Item Learner Trained Daily and the Window-based Learner Trained Once

Chapter 5

Conclusion

5.1 Contributions

This dissertation project extends the popular sequential next item learner, SASRec [14], to train on window-based objectives. We implement the all action and dense all action learning introduced in [24] as well as two new objectives, super-dense all action and target-dense all action. The results show that these window-objectives are better suited to make prediction over longer future temporal periods (future windows).

The main contribution of this project is combined learning which leverages both sequential next item and window-based learning. Combined models achieve higher Top-N performance than pure window-based models on the future window.

In addition, the combined learning hybrid model shows the SOTA performance. This is because the hybrid model can control the amount of emphasis put on the two learning ideas. It goes to show that there is benefit in having control over the contribution of each learning paradigm. In fact, investing in tuning to find the optimal balance can maximise performance on the window prediction task.

The project also tracks the change in performance on adding time features to the various models with different learning objectives. In general, time features improve performance of prediction on the future window.

Finally, the project takes into consideration the resource implications of window-based learning in deployment scenarios. A large motivation for the Pinnerformer [24] was to remove the need to update models on daily basis by improving longer term performance. In our project, we show that the training the window-based and combined objectives once before a 14-day window yields a better performance over the window than a next item learner trained and evaluated daily over the same window. Extrapolating

this result to online learning, it means training once in 14 days with new learning ideas can yield better performance than training daily with current widely adopted learning ideas. Therefore, window-based and combined models make longer update cycles a viable deployment-level strategy. Consequently, motivating more time and resource efficient RSs.

5.2 Limitations

The work was limited by the data available for experimentation. First, only one RS related open-source dataset (ML-20M) was found that was appropriate for window experimentation offline because the requirements for studying the window prediction task are more selective: users need to have data spanning for more than the chosen window size. Second, the ML-20M was not suited for global time splits. Hence, we weren't able to create the closest imitation of an online scenario.

However, having said this, it is important to acknowledge that the Ml-20M dataset represents real-world user behaviour. In that sense this study tries to comment with user interaction patterns that could be encountered in reality.

5.3 Future Work

Directions for future research first and foremost include experimenting with more diverse data regimes. This directly addresses the limitations mentioned in the previous section. Moreover, because of limited compute only a single window size was used. It will be interesting to understand the change in performance with varying window sizes. Lastly, predictions within the window are time-agnostic, i.e., we do not care about when the actions occur within the window itself. Certain use-cases might call for prediction at specific times within the window. Including time features for items in the window then is an interesting direction of discovery.

More importantly, our work argues that learning more general ideas, through sequential learning, about the user interaction sequence boosts performance on the window-prediction task. This echoes findings from Natural Language Processing (NLP) that show that pretraining on generative tasks creates a general language understanding that improves performance when finetuning on discriminative tasks [26]. The huge benefit of the generative pretraining is that it is unsupervised and does not require annotated

data. Hence, it can be trained on a large corpus of unlabeled language data. On the other hand the discriminative tasks are limited by the availability of labelled datasets.

Similarly, training on windows is limited by more selective data requirements (see above) while we can still proceed to learn general sequential ideas on larger datasets. We do something like this in Section 4.3; however, this is a limited investigation.

We suspect that there is a huge scope of research in introducing modern pretraining-finetuning ideas from NLP in the SRS domain. Future works can be geared towards experimenting with new ways to develop general sequential understanding or with how current SOTA techniques transfer to window-based learning. For example, the cloze task as utilised in the BERT4REC [32] can be used for pretraining.

Bibliography

[1] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 108–116. ACM, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[3] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In Paolo Cremonesi, Francesco Ricci, Shlomo Berkovsky, and Alexander Tuzhilin, editors, *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, pages 152–160. ACM, 2017.

[4] Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, 2010.

[5] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations. *ACM Trans. Inf. Syst.*, 39(1), 11 2020.

[6] Umer Gupta. Evaluating the window-based approach for sequential recommendation in data dense and data sparse environments: A study, 2023. University Of Edinburgh, Informatics Master's Dissertation Project Proposal.

[7] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[8] Ruining He, Wang-Cheng Kang, and Julian J. McAuley. Translation-based recommendation. In Paolo Cremonesi, Francesco Ricci, Shlomo Berkovsky, and Alexander Tuzhilin, editors, *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, pages 161–169. ACM, 2017.

[9] Ruining He and Julian J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 191–200. IEEE Computer Society, 2016.

[10] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang, editors, *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 843–852. ACM, 2018.

[11] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[12] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz, editors, *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 505–514. ACM, 2018.

[13] J. F. James. *A Student's Guide to Fourier Transforms: With Applications in Physics and Engineering*. Student's Guides. Cambridge University Press, 3 edition, 2011.

[14] Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 197–206. IEEE Computer Society, 2018.

[15] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus A. Brubaker. Time2vec: Learning a vector representation of time. *CoRR*, abs/1907.05321, 2019.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[17] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[18] Yehuda Koren, Steffen Rendle, and Robert M. Bell. Advances in collaborative filtering. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 91–142. Springer US, 2022.

[19] Jiacheng Li, Yujie Wang, and Julian McAuley. Time Interval Aware Self-Attention for Sequential Recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 322–330, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In Ee-Peng Lim, Marianne Winslett, Mark Sanderson, Ada Wai-Chee Fu, Jimeng Sun, J. Shane Culpepper, Eric Lo, Joyce C. Ho, Debora Donato, Rakesh Agrawal, Yu Zheng, Carlos Castillo, Aixin Sun, Vincent S. Tseng, and Chenliang Li, editors, *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 1419–1428. ACM, 2017.

[21] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.*, 7(1):76–80, 2003.

[22] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[23] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359, 2010.

[24] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. Pinnerformer: Sequence modeling for user representation at pinterest. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 3702–3712, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In Paolo Cremonesi, Francesco Ricci, Shlomo Berkovsky, and Alexander Tuzhilin, editors, *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, pages 130–137. ACM, 2017.

[26] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[27] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 811–820. ACM, 2010.

[28] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1257–1264. Curran Associates, Inc., 2007.

[29] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295. ACM, 2001.

[30] Guy Shani, David Heckerman, and Ronen I. Brafman. An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295, 2005.

[31] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics, 2018.

[32] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1441–1450, New York, NY, USA, 2019. Association for Computing Machinery.

[33] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 565–573. ACM, 2018.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[35] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. Contrastive learning for sequential recommendation. In *2022 IEEE 38th international conference on data engineering (ICDE)*, pages 1259–1273. IEEE, 2022.

[36] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information*

Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, pages 3320–3328, 2014.

[37] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. A dynamic recurrent model for next basket recommendation. In Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel, editors, *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 729–732. ACM, 2016.

[38] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, and Xiaofang Zhou. Feature-level Deeper Self-Attention Network for Sequential Recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4320–4326. International Joint Conferences on Artificial Intelligence Organization, 4 2019.

[39] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 1893–1902. ACM, 2020.

Appendix A

Supporting Tables and Results

A.1 Transfer Learning on Entire ML-20M

Table A.1 below presents the summary statistics of using transfer learning on the entire ML-20M dataset. That is, pretaining with next item learning is done on every user and evaluated according to the next item prediction task. On the other hand, finetuning is performed with all action learning on the subset of users with data spanning more than 14 days. Evaluation is done on the window prediction task.

Stage	Objective	Task	Data	Recall@10	NDCG@10	P90	Best Epoch	Time/Epoch (s)	Wall Clock Time (s)
Pretrain	Next Item	Next Item Prediction	Entire 20M	0.827	0.573	0.100	100	58	5807
Finetune	All Action	Window Prediction	Subsetted 20M	0.604	0.365	0.031	70	61	4294
								Total Time	10101

Table A.1

A.2 Time-Aware Hybrid Learning

Cycling Ratio	Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch	Wall Clock Time (s)
(1:1)	0.643	0.394	0.029	101	95	9569
(1:3)	0.637	0.391	0.044	95	95	9026
(1:9)	0.620	0.376	0.052	89	95	8450

Table A.2: Results for the Time-Aware Version of the Hybrid Model with Different Cycling Ratios

A.3 Window Next Item and Hybrid Models in 150 Epochs

The time-aware window next item and hybrid models do not necessarily converge in 100 epochs. We rerun them for 150 epochs to see their convergence behaviour. We see that the models find the best epoch well before 150 epochs. More importantly, there is little or no gain in performance for training after 100 epochs. Given the insignificant difference in performance, we can conclude that models more or less converge within 100 epochs.

Model	Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch	Wall Clock Time (s)
Time-Aware Window Next Item	0.630	0.381	0.053	131	110	14442
Time-Aware Hybrid	0.643	0.395	0.029	107	115	12265

Table A.3: Results for the Hybrid Objective with Different Cycling Ratios

A.4 Time Performance of Time-Aware TL Models

Learning Objective	All Action	Dense	Super-Dense	Target-Dense
Best Epoch	20	15	10	5
Wall-Clock Time (s)	2937	2608	2635	1747

Table A.4: Total Wall Clock Time for the Different Time-Aware TL Models

A.5 Next Item Model Once Before the Window

The table below summarises the results of the next item learner trained once before the window and evaluated on the window. The time-only splitting strategy is used.

Recall@10	NDCG@10	P90	Time/Epoch (s)	Best Epoch	Wall Clock Time (s)
0.512	0.289	0.036	23	30	703

Table A.5: Next Item Learner Trained Once