THE UNREASONABLE EFFECTIVENESS OF A TWO-LAYER NEURAL NETWORK FOR RECOMMENDATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a personalized recommender system using neural network for recommending products, such as eBooks, audio-books (“Anonymous audio book service”), Mobile Apps, Video and Music. It produces recommendations based on user consumption history: purchases, listens or watches. Our key contribution is to formulate recommendation problem as a model that encodes historical behavior to predict the future behavior using soft data split, combining predictor and auto-encoder models. We introduce convolutional layer for learning the importance (time decay) of the purchases depending on their purchase date and demonstrate that the shape of the time decay function can be well approximated by a parametrical function. We present offline experimental results showing that neural networks with two hidden layers can capture seasonality changes, and at the same time outperform other modeling techniques, including our recommender in production. Most importantly, we demonstrate that our model can be scaled to all digital categories. Finally, we show online A/B test results, discuss key improvements to the neural network model, and describe our production pipeline.

1 INTRODUCTION

Recently, deep learning based recommender systems gained significant attention by outperforming conventional approaches (Zhang et al., 2017). It shows promising results on products like videos (Covington et al., 2016), mobile apps (Cheng et al., 2016), music (Van den Oord et al., 2013) etc. In the papers mentioned above, we noticed that NN based recommenders are different for each product category (videos, music, mobile apps), requiring unique feature extraction methods and NN topologies. All of these challenges makes it harder to scale over different product categories. In this paper we are exploring effectiveness of a multilayer neural network (NN) for personalized recommendations of products which were never purchased before by a customer. The simplicity of this approach allows us to scale it on various categories of “Company” catalog in production. We focus on improving accuracy of the neural network based personalized recommender.

It is noticed in (Covington et al., 2016) that accuracy of NN depends on how the problem is formulated. They found that NN performs better when it is trained to predict the user’s next purchase, rather than a set of randomly held-out purchases. We use the same idea, but on top of this, we propose to train NN model to predict not only future purchase, but all future purchases in the certain time (for example in the next week).

Capturing temporal popularity (trendiness) also called seasonal changes of consumption pattern is a challenging and important problem in recommender systems (Wu et al., 2017), which can impact the accuracy of the model over time. In (Wu et al., 2017; Ko et al., 2016) authors propose methods to capture seasonality changes using sequence modeling. Another approach (Song et al., 2016) models both long-term static and short-term temporal user preferences. In both cases they use different versions of recurrent neural networks. In this paper we propose to combine predictor model (which can captures short term preferences and recommend products which are currently popular) with auto-encoder model (which can capture static customer preferences and recommend products which were popular at any time in the past) using feed forward NN. These models are combined by training them jointly. We re-train NN model every day to learn new popular products and changes in customer preferences. Even though our model is simpler then RNN, we show that it captures seasonality changes well.
Improving NN based recommender is an important problem, for example in (Covington et al., 2016) authors observed that adding features and depth significantly improves precision on holdout data from YouTube catalog. In (Cheng et al., 2016) authors show that wide and deep NN with multiple features can improve performance of the neural network on mobile apps. So both methods (Cheng et al., 2016) and (Covington et al., 2016) require different production pipelines for different data sets: video and mobile apps. In this paper we use only purchase history and focus on improvements of NN accuracy by applying different splits of the training data. It simplifies the production pipeline and allows us to reuse it on all digital categories: video, eBooks, audio-books, mobile apps, and music.

Another way of improving recommender system is time decay, which was introduced by (Xia et al., 2010) for collaborative filtering. We also apply it on input data for the neural network based recommender and observe positive impact on accuracy metrics. Our contribution is to use convolutional layer (LeCun & Bengio, 1998) for estimating the shape of time decay function. Convolutional layers are used in existing recommender systems, but it is applied for different purposes, for example in (Hsu et al., 2016) convolutional layer is used for learning local relation between adjacent songs, in (Zheng et al., 2017; Kim et al., 2016) it is used for text feature extraction and in (Van den Oord et al., 2013) it is used for extracting features from audio signal.

There can be millions of products in the catalog and it is a hard problem to run NN based recommender in production with such amount of items (Covington et al., 2016). Both (Covington et al., 2016) and (Cheng et al., 2016) are splitting the problem into candidate generation and ranking. Candidate generation retrieves a small subset of products from a large corpus. These candidates should be relevant to customer interest. Ranking does a fine-level scoring of the candidates and in addition to consumption history it can use more features (context, impression, etc). Another way of scaling this problem is to learn similarity between products using DSSM approach (Elkahky et al., 2015) which is relying on content features. In this paper we focused on training end-to-end one neural network which is using only purchases events. On one hand it simplifies the production pipeline, because there is no splitting into candidate generation and ranking models and there is no special feature extraction step. But on the other hand we have to deal with large dimensionality of input features and labels. To solve this problem we use multi-GPU model parallelization, implemented by our team in “Anonymous” library (10). It allows us to re-train large NN models every day and produce fresh recommendations for our customers.

In this paper, we are focused on modeling consumption patterns in digital products (For example, recommending movies to customers based on the movies already purchased). Depending on the domain, we also exclude movies that were already purchased by the customer while computing offline metrics as well as recommending online. We present different methods of splitting the training data and observed that it can improve NN based recommender accuracy metrics. Techniques like the one presented here feed into recommendation technology deployed at “Company”.

The rest of the paper is organized as follows. Section 2 introduces offline metrics used for algorithm evaluation. Section 3 details our NN model development procedure, including how different methods are compared. Section 4 provides extensive offline evaluation results, in conjunction with model property exploration. Section 5 presents how to run NN model in production and shows on line A/B test results. Finally, section 6 presents our conclusions.

## 2 Offline Evaluation Methodology

There are many different metrics focusing on specific properties of the recommendation algorithm (6, 2014). Among all, root mean square error (RMSE) is the most popular one ((Qu et al., 2016), (Sedhain et al. 2015)). It requires explicit feedbacks (ratings). Nevertheless, in many practical applications recommender systems need to be centered on implicit feedbacks (Hu et al., 2008). Implicit information like clicks and purchases are normally tracked automatically, customers do not need to explicitly express their attitude, therefore are easier to collect. In the scenario of predicting future purchase from implicit feedback data, we use two metrics throughout evaluations in this paper: Precision at K and Product Converted Coverage (PCC) at K.
Precision at K is the accuracy of the predicted recommendations with respect to the actual purchases:

\[
Precision@K = \frac{1}{C} \sum_{i=0}^{C-1} \frac{|\{P_i\} \cap \{T_i\}|}{K},
\]

where \( K \) is the position/rank of a recommendation, \( i \) is the customer index, \( P_i \) is top \( K \) recommended items for customer \( i \), \( T_i \) is actual consumptions for customer \( i \) represented as the set of items the customer purchased in the evaluation period (where interaction can be purchases, watches, listens), \(|P|\) is the number of items in set \( P \), \( P \cap T \) is the intersection between sets \( P \) and \( T \), and \( C \) is the number of customers.

While having high precision is necessary, it is not sufficient. A personalized recommender should also recommend diverse set of items (Adomavicius & Kwon, 2012). For example, if precision is high with no diversity, then recommendations looks like a hall of mirrors showing only products in a single topic. Therefore, to guarantee the diversity of recommendations, we use products converted coverage at K. It captures the number of unique products being recommended at top K and at the same time purchased:

\[
PCC@K = \frac{1}{D} |\bigcup_{i=0}^{C-1} (\{P_i\} \cap \{T_i\})|,
\]

where \( \bigcup_{i=0}^{C-1} (X_i) \) represents union of sets \( X_0, X_1 \ldots X_{C-1} \), \( D \) is total number of products.

Using held-out labels to measure a recommender’s efficacy is leaking future purchase information (Covington et al., 2016). Consequently, there exists the risk of having inconsistent performance between offline and online evaluation. In order to reduce this gap and emulate real production environment, the test metrics in this paper are measured on future purchases instead of held out data.

3 Offline Model Development

3.1 Data Pre-processing

Apart from metric selection, one general issue related to a recommender system is how to reduce the gap between offline and online A/B test results. With the intention of using historical consumptions for training and future purchases for testing, we split data as shown in Figure 1. Users’ consumption history is divided into past and future parts by pre-selected date (\( dyz \)), after which offline accuracy metrics are measured on future part. Each purchase event of a particular customer is represented by one-hot encoding in vectors: \( X, Y, Z \), as shown in Figure 1 where: \( X \) – training data input; \( Y \) – training data output; \( XY \) – testing data input (concatenation of \( X \) and \( Y \)); \( Z \) – testing data output; \( dx, dxy, dyz \) – minimum and maximum dates in training data input \( X \); \( dz, dz \) – minimum and maximum dates of purchase in testing data output \( Z \). They are selected so that \( dz - dyz \) is between 1 week and 2 weeks.

Figure 1: Purchase history data split. Consumption history is divided into three pieces \( X, Y \) and \( Z \), by date split \( dx, dxy, dyz \) and \( dz \). \( P_v \) is product \( v \) purchased at YYYY/MM/DD. For example, \( P_1 \) – product 1 purchased at 2012/11/15, \( P_2 \) – product 2 purchased at 2013/06/21.

In practice, \( dxy \) is selected so that \( dyz - dxy \) is between 1 week and 3 months. By doing this, neural network will capture seasonality change. \( dx \) is selected so that \( dxy - dx \) is less than 2 years. During training we consider \( X \) as past purchases and our goal is to predict future purchases \( Y \). During testing we concatenate \( X \) and \( Y \) data and \( XY \) is used to predict \( Z \).

Purchase history is inherently noisy (Hu et al., 2008). It has different kinds of noises. For example, customers with a small number of purchases (which may not be reflected of their true taste due to the
small sample), customers with a lot of purchases (which might indicate a company or bulk account). To deal with this, we set the minimum and maximum number of purchases and exclude customers outside of these limits.

We apply the same data split (Figure 1) for extracting validation data sets which are used for hyperparameters optimization. Validation data for \(X, Y, Z\) are extracted using date thresholds shifted by one month \(dx - \text{month}, dxy - \text{month}, dyz - \text{month}, dz - \text{month}\). Then we train models with different hyper parameters on \(X\) and \(Y\) and pick the one which has highest accuracy on \(Z\).

During offline evaluation, we limit number of products to order of \(10^5\) and reduced number of testing samples to order of \(10^6\). It is worth noting that the purchase data has two dominating properties. First, the distribution is long-tailed, more than 90\% of purchases are covered by less than 20\% of \(X\). This fact allows us to reduce dimensionality of recommended products by four times with minimum influence on precision, but PCC can be impacted. Second, the purchase history is sparse. On average customers have purchased less then 1\% of all products in the input data \(X\). For efficient model training, we use sparse matrix multiplication in our production pipeline.

3.2 NEURAL NETWORK BASED RECOMMENDER – PREDICTOR MODEL

It is important to learn what customers are interested in buying now. For example, given one customer’s purchase history in the last two years, we aim to predict their next purchases in the next week. We use NN based approaches because they are showing promising results (Covington et al., 2016; Cheng et al., 2016).

The recommender model for \(L\) layers is defined by the function at each layer, and the loss function, as shown in Figure 2a, and illustrated below using the equations corresponding to each layer:

\[
A_0 = X + \xi; \quad A_j = \text{Drop}(\text{Act}(W_j \cdot A_{j-1} + B_j)); \quad \hat{Y} = \text{Sigm}(W_L \cdot A_{L-1} + B_L);
\]

\[
\text{loss} = \text{Cost}(Y, \hat{Y}) + \lambda \sum_{i=1}^{L} |W_i|^2 + \beta \sum_{i=1}^{L-1} KL(\rho_i, \rho'_i);
\]

where weighted cross-entropy is used as the cost function:

\[
\text{Cost}(Y, \hat{Y}) = -\sum_{i=0}^{D-1} (w_1 \cdot y_i \cdot \log(\hat{y}_i) + w_0 \cdot (1 - y_i) \cdot \log(1 - \hat{y}_i))
\]

with, \(X, Y\) – training input and output data in Figure 1; \(X, Y \in \{0, 1\}^D, D:\) number of recommended products; \(\hat{Y}\) – output scores of the NN; \(L:\) number of layers in the NN; \(W_j, B_j:\) weight matrix and bias to be learnt; \(j = 1, \ldots, L; A_j:\) is the activation at layer \(j; \text{Act}(\ast)\) – the nonlinear activation function (Relu, Sigmoid, (Nair & Hinton, 2010) etc.); \(\text{Drop}(\ast)\) – drop out function (Srivastava et al., 2014); \(\xi:\) – noise added to input data, for increasing robustness of the model (Vincent et al., 2008); \(w_1, w_0:\) – constant weights for purchased and non-purchased products respectively, introduced for balancing number of purchased vs non-purchased products;
\[ KL(\rho, \rho_i') = \rho \cdot \log (\frac{\rho}{\rho_i}) + (1 - \rho) \cdot \log (\frac{1 - \rho}{1 - \rho_i'}) \] - penalty function \cite{Ng2011} on hidden and output nodes to force the network to only activate some percentage of the hidden and output nodes; \( \rho \) – sparsity parameter; \( \rho_i' \) – average activation of unit \( i \) \cite{Ng2011}.

Parameters of the noise \( \xi \), drop-out function, number of layers \( L \), set of activation function \( \text{Act}(\ast) \), weights \( w_1, w_0, \lambda \) and \( \beta \) are hyper-parameters of the neural network model \( \text{NN} \), which can be tuned via hyper-parameter search on validation data. We also explore learning rate, momentum, mini batch size, number of epochs, type of the optimizer (SGD, Nesterov \cite{Nesterov1983}, etc). We tested deeper networks on our data sets but did not observe significant improvements. In all the experiments mentioned in this paper, our models used neural networks with two hidden layers.

During evaluation we feed \( XY \) data to the NN model and produce output scores \( \hat{Y} \). Then these scores are sorted and the top \( K \) products are returned as recommendations. Before sorting, all previous purchases (products belonging to data \( XY \) ) of the selected customer are removed from the recommendations, so that only new products are recommended. During evaluation, these recommendations are compared with data \( Z \) for accuracy calculation.

The data split method used for training model \( \text{NN} \) via the multi-label classification is shown in Figure 1. We call it \textit{predictor} model because it predicts future purchases \( Y \) given past purchases \( X \). This model can produce personalized recommendations of products which were popular during the last week. As a result we can expect high precision of predicting future purchases, but PCC will be constrained by the products which were purchased in the last week \( (Y) \) only.

3.3 Offline Comparison of Predictor Model with Other Approaches

We check the effectiveness of \textit{predictor} model by comparing it against the following models:

- **FISM** \cite{Kabbur2013} is based on item-to-item similarity matrix factorization, in which two low dimensional latent factor matrices are learned. Bayesian Personalized Ranking (BPR) \cite{Rendle2009} loss with negative sampling is used. The model is tuned for different values of rank \( k \) and learning rate \( l \).
- **Fossil** \cite{He2016} combines similarity matrix factorization which utilizes Markov chains to capture temporal dynamics. In addition to rank and learning rate, we also tune for \( L_2 \) regularizer and order of the Markov chains.
- **LSTM** is commonly used for dealing with sequence of items \cite{Graves2013}. We use the unidirectional LSTM with a softmax output layer to choose top \( k \) recommendations, as implemented in \cite{Devooght2016}. Best result is selected upon different layer sizes \{32, 64\}, batch sizes \{16, 32\} and size of an extra embedding layer before the recurrent layer \{0, 32, 128\}.
- **GRU** Similar to LSTM, GRU is also a gated RNN in which information flows are controlled, but with a different gated recurrent unit \cite{Hidasi2015}. The model is tuned in same manner as LSTM.
- **FastXML** Our recommender system can be formulated as an extreme multi-label classification problem. Therefore, we compare our approach with Fast and Accurate Tree Extreme Multi-label Classifier \cite{Prabhu2014}.

On Figure 3 we see that our model \( \text{NN} \) has the best Precision, and our production CF has the best PCC. The next best model is FastXML. In the rest of this paper, we will be focusing on improving the NN model in terms of Precision and PCC.

Figure 3: Accuracy metrics of different approaches on “anonymous video service” purchase history

\footnote{FastXML is implemented by \url{https://github.com/Referer/fastxml} all the rest is implemented by \url{https://github.com/rdevooght/sequence-based-recommendations} \cite{Devooght2016}}
3.4 LEARNING TIME DECAY FUNCTION WITH CONVOLUTIONAL LAYER

One of the ways to improve a recommender is by using time decay function (Xia et al., 2010), so that the importance of recent purchases can be increased. There are multiple parametrical functions that can be used for this purpose but we do not know which one of them is the best. That is why we propose to use a NN with convolutional layer that can learn the shape of the time decay function. NN model with convolutional layer is described below and shown on Figure 2b.

Date of purchase $XT$ is processed by $Bin()$ function, it converts purchase date into 32 binary encodings. For example, recent purchase will be represented as $[1, 0, \ldots, 0]$, whereas purchase which was done 2 years ago will be encoded as $[0, \ldots, 0, 1]$. Two years of purchase events are divided into 32 buckets, so that 22 days belongs to one bucket. All binary encoded data are concatenated into one vector $V$, which will be used by one dimensional convolutional layer (with stride 32).

$$V = Bin(XT);$$

$$A_0 = Conv(V) = Act(\sum_{i=0}^{31}(V_{(i+32) \cdot d} \cdot w_f + w_b)),$$

where $w_f$ is the one dimensional convolutional kernel (with size 32), and $w_b$ is the bias for $d = 0 \ldots D - 1$. The remaining functions are the same with (3).

To see how convolutional layer can explore the shape of time decay, we use a parametric function as a baseline. It is defined as:

$$Td(d, \text{decay}) = \begin{cases} 
1/(1 + \frac{\text{ref} - d}{\text{decay}}), & d < \text{ref} \\
0, & \text{else}
\end{cases}$$

where $d$ – input day; $\text{decay}$ – time decay parameter measured in days; $\text{ref}$ – reference day, which is equal to $\text{dxy}$ during training and $\text{dyz}$ while generating recommendations.

We trained three models, the first model was a NN predictor defined by (3), the second used a convolutional layer defined by (4) and shown in Figure 2b and the third one took dates of input purchases decayed by (5) and then fed this data to the input of model (3). The time decay parameter of the third model was learnt using hyper-parameter optimization on validation data, as described in section 3.1. The optimal value for this parameter was found to be 40. These models were trained on “anonymous video service” data sets with different date splits: October data set ($dx = 2012/10/31, dxy = 2014/09/23, dyz = 2014/10/01, dz = 2014/10/08$) and Christmas data set ($dx = 2011/10/31, dxy = 2013/12/15, dyz = 2013/12/22, dz = 2013/12/29$). We selected Christmas week to see that time decay function with convolutional layer will have a spike at Christmas week one year in the past.

The shape of time decay function learnt by convolutional layer is shown on Figure 4. We observed that the function modeled on October data (date split $dyz = 2014/10/01$) is going down with increasing $d$ (red dashed line). So it can be well approximated by the parametrical function (5) (green dashed line).

For modeling seasonal activities like movies popular during Christmas every year, one would expect movies that were popular last Christmas week will be popular this Christmas week as well. In our experiment, as we expect, the convolutional layer learns a different shape of the time decay function for the Christmas week ($dyz = 2013/12/22$, blue solid curve). It is demonstrated by a local spike
observed during that time, highlighting the popularity of Christmas movies at the same time of the year. This spike is located at Christmas week of the previous year.

Table 1 shows the results that compare predictor model with and without time decay. Time decay on input data slightly improves precision, but PCC is reduced. We demonstrate that the shape of the time decay function learned by the convolutional layer can be well approximated by the parametrical function (6). Given that, and for production pipeline efficiency, we use parametrical function (6) as an approximation of time decay shape in rest of this paper. But the idea of integrating a convolutional layer into the architecture can be adjusted to more general framework of NN-based recommenders.

Table 1: Predictor models with different time-decay

<table>
<thead>
<tr>
<th>Predictor models</th>
<th>Precision@1</th>
<th>PCC@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default model (3)</td>
<td>0.072</td>
<td>0.018</td>
</tr>
<tr>
<td>with conv time decay (4)</td>
<td>0.074</td>
<td>0.016</td>
</tr>
<tr>
<td>with parametrical time decay (5)</td>
<td>0.075</td>
<td>0.017</td>
</tr>
</tbody>
</table>

3.5 NN BASED RECOMMENDER WITH COMBINED PREDICTOR AND AUTO-ENCODER MODELS

On Figure 3 we observe that predictor model has highest precision@K but lower PCC@K. The reason for lower PCC lies in the method of splitting the training data as we train the model to predict more recent purchases from the past week. As a result NN model cannot predict products which were purchased only one month ago because they were never seen in training output data $Y$. Predictor model is recommending personalized popular products. It captures short term preferences and recommend products which are popular now. One of the methods of increasing PCC is to use auto-encoder neural network (Hinton & Salakhutdinov, 2006), which learns representations of the input data $X$ by learning to predict itself in the output. By training the representation and reconstructing all of a user’s purchases, it can capture static customer preferences and predict products purchased at any time in the past. As a result, we can expect an increase in PCC in auto-encoder compared to predictor model since the diversity of products over a two year period learned by an auto-encoder is more than a one week period learned by a predictor.

In light of this, we introduce a hybrid approach which combines predictor and auto-encoder models. We show that they can increase both precision and PCC, or increase one of them and do not change another one.

The combined model may be illustrated by the equations describing the per-layer functions and shown on Figure 2c

$$A_0 = Td(XT, ti) + \xi; \quad A_j = Drop(Act(W_j \cdot A_{j-1} + B_j)); \quad \hat{Y} = Sigm(W_L \cdot A_{L-1} + B_L);$$

$$\text{loss} = \text{Cost}(XT, X, \hat{Y}) + \lambda \cdot \sum_{i=1}^{L} |W_i|^2 + \beta \cdot \sum_{i=1}^{L-1} KL(\rho, \rho');$$

$$\text{Cost}(XT, X, \hat{Y}) = \text{auto}(XT, X, \hat{Y}) + \text{predict}(Y, \hat{Y}),$$

in which

$$\text{auto}(XT, X, \hat{Y}) = - \sum_{i=0}^{D-1} \{wa_1 \cdot Td(xt_i, to) \cdot \log(\hat{y}_i) + wa_0(1 - x_i) \cdot \log(1 - \hat{y}_i)\},$$

$$\text{predict}(Y, \hat{Y}) = - \sum_{i=0}^{D-1} \{wp_1 \cdot y_i \cdot \log(\hat{y}_i) + wp_0 \cdot (1 - y_i) \cdot \log(1 - \hat{y}_i)\}.$$  

Here, we have $wa_1, wa_0$ – constant weights for purchased and non-purchased products respectively, set to be 1 by default. They introduced for balancing number of purchased vs non-purchased products, these weights are used for auto-encoder model; $ti$ – time decay parameter applied on input data; $to$ – time decay parameter applied on output data in cost function; $XT$ – purchase dates of products $X$. $xt_i$ is scalar value of vector $XT$; $Td(d, decay)$ – time decay function defined in (5)
applied on input data and cost function of auto-encoder; \( d \) – purchase date; \( \text{decay} \) – time decay hyper-parameter measured in days. The remaining variables are as in model (3).

Auto-encoder (8) and predictor (9) models are trained jointly using mixed cost function (7). Each output label \( X \) in the auto-encoder is weighted by the function \( Td() \), which is controlled by decay parameter. It controls the impact of auto-encoder model. We selected function (8) for controlling the impact of auto-encoder. We also apply time decay on the input data \( XT \) where shape of time decay function is defined by parametrical definition (5) or can be learned using a conv layer (4).

Model (7) can be interpreted as multi-task learning (Caruana, 1998), where loss for predicting past purchases (8) is task one, and loss for predicting future purchases (9) is task two. Model (6) is a generalized version of model (3) and by adjusting \( ti \) and \( to \) we can get predictor model (9) or a mixture of predictor and auto-encoder models.

\[
\text{Predictor} \quad \text{Equally Weighted (EW)} \quad \text{Soft split}
\]

(a) Predict \(\text{r} \) (b) Equally Weighted (EW) (c) Soft split

Figure 5: Different weighted combinations of predictor and auto-encoder models

Predictor model is defined by fixed parameters, \( ti = 10^6, to = 10^{-6} \), \( wa_1 = wa_0 = 0 \) and other NN parameters were found using hyper-parameter optimization on validation data as described in section 3.1. Shape of the time decay function \( Td() \) applied on input and output data are shown on Figure 5 by red and black color respectively. Predictor model uses past consumptions \( X \) as training input data and predicts future purchases \( Y \), therefore it has higher precision and lower PCC.

Our first proposed approach is Equally Weighted (EW) auto-encoder and predictor model. It is defined by parameters: \( ti = 10^6, to = 10^6 \). Because \( ti \) and \( to \) are very large, there is no time decay applied. Shape of the time decay functions applied on input data and cost function (8) is shown on Figure 5. It uses past purchases \( X \) as training input data and predicts both past \( X \) and future purchases \( Y \). We can expect higher PCC in comparison to predictor model because auto-encoder and predictor model can predict products which belong to both input \( X \) and output \( Y \) data sets.

Another approach is to use time decay weighted auto-encoder and predictor model (soft split). It is defined by parameters: \( ti = 40, to = 4 \), which were found using hyper-parameter optimization. It applies time decay on input data \( X \) to capture the importance of recent purchases. Its shape is shown in Figure 5 by red color. This model predicts both past and future purchases, but the importance of predicting past purchases is weighted by function \( Td() \) highlighted by the black curve in Figure 5. Therefore, it will allow to keep properties of predictor model (for example high precision) and at the same time have properties of EW model (for example, higher PCC). We call this model soft split because the training output data has time decayed training input data included in it, compared to the hard split of predictor model based on a fixed date split.

4 EXPERIMENTS

We evaluate EW and soft split NN models on “anonymous video service” dataset. Then we present our model’s scalability to work on different digital product categories, and explicate the learning pattern along training process. We also demonstrate the model’s ability to capture seasonality effects.

4.1 METRICS ON “ANONYMOUS VIDEO SERVICE” (AIV)

As observed in Figure 3 predictor model has high precision against other methods, but comparatively low PCC. We trained both EW and soft split NN model to check how our proposed model is impacting accuracy metrics.
We select predictor model and production collaborative filtering from Figure 3 to serve as baselines, other approaches are excluded because NN is already outperforming. All pre-processed data and model metrics remain unchanged. Results are shown in Figure 6a. We observe that predictor model has higher precision and lower PCC than other methods. The model with soft split has slightly lower precision, but at the same time PCC is boosted to be twice. EW model increases diversity of recommendations by two times, but at the same time precision was reduced significantly. That in return explains our intuition of combing predictor and auto-encoder models in a soft way.

![Figure 6: Accuracy metrics on different categories](image)

### 4.2 Model Scalability

It is important for recommendation algorithms to be easily operated across different categories, especially at “Company” scale. In this section, we present how our approach can be trained on all categories of digital products without additional feature engineering.

As expected, auto-encoder’s ability to model long term static user preference allows us to increase PCC. Model with soft data split has higher precision in comparison with other approaches and higher PCC than predictor model. But PCC is comparable or lower than EW model (Figure 6).
Production collaborative filtering has higher PCC than any other approaches (Figure 6). One of the reasons why NN based recommender has lower PCC is due to input/output dimensionality reduction that we apply in production for daily model re-training.

We show that by applying the learning process of proposed model to different categories dataset, we can gather information from those different patterns. For example, predictor approach is modeling short-term user preferences: given two years history of past purchases it predicts future purchases in the next week. This approach works well on popularity driven categories like video, in which customers are biased more to purchasing popular video products. Meanwhile, there other categories which have different consumption pattern and predictor model is not the best. For example, we observe that predictor model on video data (Figure 6a) has much higher precision than production collaborative filtering. Predictor model is producing personalized recommendations which are popular now. In contrast, on eBooks and audio-books (Figure 6e, 6c) we observe that predictor model has the same precision with production collaborative filtering, it indicates that consumption pattern of these two categories is different with video data set (where predictor has the highest precision). In both eBooks and “Anonymous audio book service” model with soft split has the highest accuracy, so there is a consumption pattern similarity between both of these categories. On Music and Apps dataset, soft split has significantly higher Precision and higher PCC than predictor model.

We showed that by combining predictor and auto-encoder models we can achieve significant accuracy improvements. The simplicity of the model allows us to scale it in production on all digital categories.

**Figure 7**: Properties of predictor model on AIV data. (a)(left) Convergence of metrics over epochs. (b)(right) Metrics dependent on minimum number of purchases

### 4.3 Model properties

To understand the strength and weakness of our model, we explore the convergence properties in NN model training process. Here we demonstrate how Precision and PCC change along epochs, together with how they are affected by number of purchases. From Figure 7 we observe that precision converged after 10 epochs, whereas PCC is converging after 70 epochs. NN learns most popular products and reaches high precision fast. With more iterations precision stays stable, but PCC increases significantly as NN learns to predict less popular products or the tail of the products distribution (PCC is increased by 3 times after 70 epochs).

It is observed in Figure 7 that the model accuracy depends on number of purchases customer has. Recommendations will be more popularity biased if customer is inactive and has few purchase, therefore as a result, becomes less accurate. As shown in the figure, Precision@1 and PCC@1 are low on customers who has number of purchases $p \leq 3$, but for customer who has a lot of purchases, recommendations will be more accurate and more diverse: Precision@1 and PCC@1 are increasing with growth of minimum number of purchases $p$.

Influence of different number of layers are also studied. We noticed that deeper NN models did not improve accuracy significantly and two hidden layers was enough. This is not surprising as we do not use more complex feature (product description, customer features etc) as in (Covington et al., 2016). Number of hidden units in hidden layer was important for increasing accuracy metrics, thus we did hyper parameters optimization with number of hidden units in range 128, ..., 8000. On the most of the categories, NN with ReLu activation has higher accuracy metrics. This result is consistent with (Covington et al., 2016).
4.4 Seasonality Evaluation

To ensure that our model captures seasonality changes or trendiness, we re-train it every day and test how it reacts to seasonal effects. This is validated by comparing the most popular products produced by NN with the most popular products which are actually purchased by customer at different time. We trained and evaluated NN model on two data sets: October and Christmas data sets, as described earlier in Section 3.4.

In Table 2, we show the most popular products recommended by our model for each date set on left side, and as a contrast, the most purchased ones on the right side. The overlaps between the most recommended and the most purchased products are highlighted in bold. By training the model every day and targeting on future purchases, it successfully captures temporal dynamics. For example, the old movie “National Lampoon’s Christmas Vacation” is recommended most during Christmas season and at the same it was the most purchased one. So our model is able capture the trendiness of movies that are watched during Christmas.

Table 2: Top six the most recommended and the most popular videos at different dates

<table>
<thead>
<tr>
<th>Most recommended in October</th>
<th>Most purchased in October</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Most recommended in Christmas</th>
<th>Most purchased in Christmas</th>
</tr>
</thead>
</table>

We also measure accuracy metrics to validate the performance of NN models quantitatively. Precision@K and PCC@K are calculated in October and Christmas week and shown on Figure 8. As can be seen, Christmas recommendations has higher Precision@K and lower PCC@K, in comparison with October recommendations. There is an acceptable temporal variation in the accuracy metrics, so we can confirm that out NN model is adapting to seasonality changes. At the time of this test our model was not in production yet, so our recommendations did not impact customer consumption pattern.

In spite of these promising offline results, we must recognize that these algorithms are evaluated against historical data. That data is based on customer consumption history influenced by collaborative filtering algorithms running in production. Therefore, online A/B tests are conducted and discussed in next section.

5 Running NN models in production and online A/B test results

One of the pipeline requirements is to produce fresh NN model with recommendations every day. Another one is to scale this approach to all digital products. For this reason, the designed NN pipeline as shown in Figure 9 consists of several steps: data pre-processing, model training, recommendations generation and serving database of recommendations. It allows us to train multiple NN models for different categories offline, then generate recommendations for all customers offline and store it into database, and in the end serve multiple databases online for all digital products.
We re-train NN model every day and generate recommendations for all customers belonging to a selected category daily. This approach will produce different recommendations every day even if particular customer has not bought anything recently. Daily re-training also helps to learn new items which become popular at that time. Date splits, described in section 3.1, are updated every day so that $d_{yz} = \text{current day}$, $d_{xy} = d_{yz} - \text{one week}$, $d_x = d_{yz} - \text{two years}$. We feed $XY$ data to the input of the NN model and produce output scores $\hat{Y}$. These scores are sorted and the top $K$ products with the highest scores are returned as recommended products. Before sorting, all previous purchases (products belonging to data $XY$) of the selected customer are removed from the recommendations so that only products which customer has never purchased before are recommended.

The first step of the pipeline is data pre-processing, described in Section 3. For this we use Spark on multi host map-reduce cluster. Training time can be impacted by number of training samples, so we control it by keeping number of training samples constant and by removing noisy samples. Another factor impacting training time is the model size which is defined by number of recommended products and number of units in hidden layer. We control number of recommended products by clipping the tail of the product distribution. The next step is neural network model training. Training time is governed by the size of the weight matrices $W$. Bigger size of model takes more time to train. We use multi-GPU model parallelization which we implemented in the “Anonymous” library (10).

In order to scale up and compute the predictions daily we do data parallel predictions. We split data into batches of customers and run predictions through Spark on a GPU cluster (11). Each predictor receives customer purchase history and the trained NN model generates recommendations (predictions) which are stored in database. The online recommender service reads these predictions and displays them to customers.

Above steps are repeated every day, so that NN model uses the freshest purchase history and can capture seasonality changes.

Figure 9: Production pipeline

Our pipeline is similar with (Cheng et al., 2016), the main difference is that we pre-generate recommendations offline. As a result, our online recommender remains simple and supports multiple categories with no modifications. Also Data pre-processing, Model training and Recommendations generations pipeline can be used by multiple category recommenders sequentially or in parallel if there is an extra hardware available. The side effect of our approach is offline recommendations generation is delaying the propagation of recent purchases into our model.

We conduct online A/B tests of the proposed NN based recommender on several digital categories. The length of the test is four weeks. Fifty percent of all customers in the selected category are exposed to new recommendations and remaining fifty percent to existing recommendations. We observe p-value to be less than 0.05 for all categories. During experiment we control only one widget. There are also other widgets which belong to the same category but producing different kind of recommendations, for example repeated purchases, new releases, discounts, etc. All of these widgets will impact Key Performance Indicator (KPI) metrics which are measured on the whole category. That is why even though we observe increase in number of purchases for a tested widget by more than 30%, overall impact on the whole category is lower, because other widgets belonging
to the same category are impacted. Depending on the category our KPI is the number of purchases or minutes streamed.

NN based Mobile Apps recommender increased the number of purchases by 1.25% on Apps Storefront (“Company” digital devices). We observed 0.73% increase in “anonymous video service” minutes streamed on “Company web page”. NN based eBooks recommender increased number of purchases by 0.79% on “Anonymous ebook reader” devices. NN based audio books recommender increased number of purchases by 1.5% on “Anonymous audio book service” desktop, tablet and mobile devices.

6 CONCLUSION AND FUTURE PLANS

We described a personalized neural network based recommender system which is currently used by “Company” for digital product recommendations: eBooks, “Anonymous audio book service”, Apps and Video. We are currently working on expanding it to non-digital categories. We showed that splitting customer purchases into a history period (input) and a future period (output) in our models led to good results, and some of our production models use this approach (with soft split which combines the auto-encoder model with the future predictor model). We have applied time decay learnt by convolutional layer, or defined by parametrical function to consumption event. It captures the importance of recent activity, and combined with soft split, it leads to significant improvements in offline metrics. We demonstrated that two layer neural networks are outperforming other NN based approaches which are more complicated than our method. Because of simplicity of the NN model we can scale it on all digital categories. We observed significant KPI improvements during online A/B tests in production.

REFERENCES

Anonymous DL library. Anonymous URL.

Anonymous blog. Anonymous URL.


