Hybrid Hyper-Parameter Optimization for Collaborative Filtering

Péter Szabó and Béla Genge

August 31, 2020

Hybrid Hyper-parameter Optimization for Collaborative Filtering

Péter Szabó
Department of Electrical Engineering and Information Technology
University of Medicine, Pharmacy, Science and Technology of Târgu Mureș
Târgu Mureș, Romania
Email: peter@kaizen-ux.com

Béla Genge
Department of Electrical Engineering and Information Technology
University of Medicine, Pharmacy, Science and Technology of Târgu Mureș
Târgu Mureș, Romania
Email: bela.genge@umfst.ro

***Abstract*—Collaborative filtering (CF) became a prevalent technique to filter objects a user might like, based on other users' reactions. The neural network based solutions for CF rely on hyper-parameters to control the learning process. This paper documents a solution for hyper-parameter optimization (HPO). We empirically prove that optimizing the hyper-parameters leads to a significant performance gain. Moreover, we show a method to streamline HPO while substantially reducing computation time. Our solution relies on the separation of hyper-parameters into two groups, predetermined and automatically optimizable parameters. By minimizing the later, we can significantly reduce the overall time needed for HPO. After an extensive experimental analysis, the method produced significantly better results than manual HPO in the context of a real-world dataset.**

***Keywords*-artificial intelligence, machine learning, recommender, collaborative filtering, hyper-parameter, hyper-parameter optimization, Adam**

I. Introduction

Predicting what the users would like to buy is crucial for e-commerce. To create those predictions, often unsupervised machine learning is used. To be able to recommend objects (items to buy) to users, the machine learning models often rely on collaborative filtering (CF) [1].

CF algorithms filter objects a user might like based on the reactions of users with similar tastes. To quantify the user's taste, we can leverage user ratings [2], [3], or we can calculate their preferences based on the recorded events. For example, we can record and quantify purchasing or viewing an object in an e-commerce shop [4], [5].

Similar to other machine learning algorithms, neural network-based CF algorithms need hyper-parameters to control the learning process. Unlike other parameters, such as node weights, hyper-parameters are not obtained via training. Moreover, most hyper-parameters cannot be inferred while the training takes place. Unfortunately, significant performance variation can be attributed to hyper-parameters, therefore optimizing those parameters is crucial to the implementation of CF algorithms [6], [7].

Hyper-parameter optimization (HPO) is the ubiquitous term for selecting and fine-tuning a tuple of hyper-parameters, with the intent of finding the optimal model. By optimal model we mean a model to minimize the loss function [8], [9].

CF research papers usually don't address the HPO issue, probably because of the time-intensive nature of traditional HPO. Dacreama, *et al.* [10] observed that, in 2019, commonly used CF datasets contain only a few hundred thousand ratings. Even for those 'tiny datasets', as Dacrema *et. al.* calls them, HPO 'can take days or weeks'.

Other authors, for example, Ebesu *et al.* [11], provide the optimal hyper-parameters for the proposed method without any indication of how those parameters are obtained. In contrast, others omit this part of the research from their published papers. For instance, Zheng *et al.* [12], documented only one set of hyper-parameters, to be used for all datasets. When a different group of researchers tried to reproduce the results using those hyper-parameters for HetRec and Amazon Instant Video datasets, the baseline algorithms outperformed the proposed SpectralCF algorithm [10].

To address the lack of hyper-parameter optimization in the literature, we provide a speed-optimized HPO method for CF, complete with a PyTorch-based implementation. The hybrid HPO solution presented in this paper can be used to find the best hyper-parameters for an Adam optimizer-based recommender system or one of the other modern stochastic gradient descent (SGD) optimizers. The solution is innovative in the sense that it requires less computation time, while producing significantly better hyper-parameters, when compared to related techniques.

The method relies on selecting the minimal amount of optimization candidate hyper-parameters and pruning trials to achieve time-efficiency. A trial is a training run on the train and validation dataset with a unique set of hyper-parameters. By pruning, we mean automatically stopping trials if the best batch result is worse than the median of batch results of previous trials at the same step. Moreover, during the experimental assessment, we empirically prove that there is no significant correlation between the number of factors and the dropout probability, two commonly used

CF hyper-parameters. Therefore, we suggest independent sampling as a better approach than relational sampling for tuning hyper-parameters.

The structure of this paper is the following: After an overview of related works in Section II, we present our hybrid approach in Section III, followed by an experimental assessment on a real-world CF dataset in Section IV. We conclude the paper with Section V.

II. Related Works

In 2019, we find several notable research papers that address the topic of HPO, mostly related to automatic HPO. Shin, *et al.* [8], proposed a stage-based execution strategy, proving that it significantly outperforms the trial-based method, using only 15% of the graphics processing unit (GPU) hours. This leads to shortening the overall training time to 23.86% of the trial-based training time. They used the CIFAR-10 tiny image dataset in their experiments. Although the image recognition problem requires different solutions than CF, their work showcases the benefits of automatic HPO, and the stage-based strategy, which served as a starting point for our research.

With the widespread adoption of machine learning, the budget constraint became apparent. Two different teams of researchers approached HPO as a hard resource constraint problem [13], [14]. Contrary to pruning the neural network (NN) size, as suggested by Lemaire, *et al.* [15], new research suggests that pruning trials seem more robust and more promising. Instead of a budget constraint, our research focuses on minimizing time given the available resources. This way, we achieve a more generic solution, which is also easier to improve and test. Instead of trying to find the best performance score on the given hardware, we try to minimize the time needed to reach a target performance score. Any improvement over our current solution needs to produce the same or better performance score under a shorter time-span on the same hardware. For obvious reasons, this kind of time reduction also leads to budget reduction. Our approach is even closer to the traditional computational time complexity-based evaluation of algorithms [16].

Akiba, *et al.* [17], introduced a new design-criteria for next-generation hyper-parameter optimization software. We note that the automatic component of the hybrid HPO presented in this paper follows similar principles. The three main principles are the define-by-run Application Programming Interface (API), which allows dynamic hyper-parameter space construction, the efficient searching and pruning strategy, and the versatile architecture.

Contrary to the researchers mentioned above, we apply HPO to collaborative filtering, instead of image recognition. While we rely on the framework created by Akiba, *et al.* [17], for the automatic search, we don't apply the procedure to all hyper-parameters. Instead, we suggest a hybrid approach to reduce the hyper-parameter space before running automatic hyper-parameter optimization, leading to a significantly reduced search time.

III. Developed Approach

The goal of the developed approach is to find the ideal hyper-parameters for the neural network (NN) trainer algorithm in order to minimize the loss function on the validation dataset. The steps of the developed approach are detailed in the remainder of this section.

A. Establishing the Ground Truth

For large datasets holdout validation is recommended [18]. While k-fold cross-validation can lead to minor accuracy improvements over holdout validation ($0.1 - 3\%$ better accuracy), the time trade-off for large datasets can be significant. As we mentioned above, our goal is to develop a time efficient approach, therefore we use a special case of holdout validation, namely, the three-way data split. The remainder of this section details the previously mentioned steps.

The three-way data split separates the tests set from the train and validation set. Only the final output of the HPO is evaluated using the test set. The validation set is used during hyper-parameter optimization. It's worth noting that for small datasets the three-way data split is not recommended, because it produces a significant variance in parameter estimations [19].

Let Y be our ground truth dataset, a sparse matrix. The rows of the matrix contain the preference of the users, and the columns the objects. As a result, $Y_{u,o}$ contains the preference of user u for object o. This preference can mean the likelihood of purchase, a given rating, or any measurement of the relationship, as long as it is positive. Zero value means that the dataset contains no information about the relationship between user u and object o. Y_{train}, Y_{val}, and Y_{test} are sparse matrices of the same shape as Y, containing the train, validation and test data-points respectively.

B. Defining the Neural Network Solver Function

To solve the problem of HPO, we define the objective function to be minimized, known as the loss function. Due to the nature of CF, there are no considerable outliers in CF datasets.

Let $\hat{Y}$ be the prediction of the algorithm about ground truth Y, and q the number of data points being predicted in the batch. The usual loss function for CF is the mean squared error (MSE). We calculate it on q data points. This is called the means squared prediction error (MSPE), defined in (1). Mean absolute error could also be used as the loss function is, but most CF data has some perturbation, and a small perturbation may have a significant impact on MAE compared to MSPE.

$$MSPE(\hat{Y}, Y) = \frac{1}{q} \sum_{a=n+1}^{n+q} (Y_a - \hat{Y}_a)^2 \quad (1)$$

Next, let $backpropagation()$ be the function responsible for backward propagation of errors [20] and let $Adam(lr, \beta_1, \beta_2)$ be the Adam optimizer. We use $Adam(lr, \beta_1, \beta_2)$ by Kingma, *et al.* [21], to minimize the loss function. This leads to the first three hyper-parameters in our hyper-parameter tuple: $lr \in (0, 1)$ denoting the learning rate, and $\beta_1, \beta_2 \in [0, 1)$ denoting the decay rates for moment estimates. It's worth noting that the Adam optimizer was chosen, because it converges much faster than AdaGrad or RMSProp algorithms, it can handle sparse gradients on noisy problems, and, most importantly, the hyper-parameter tuple to be optimized automatically will contain fewer parameters [22]. This is owed to the default values of lr, β_1 and β_2, which provide the desired results without tuning for most datasets [21].

The model is trained for a variable number of $epochs$. Let $epochs \in \mathbb{N}_{>0}$ denote the number of times the entire dataset Y_{train} is passed both forward and backward through the NN. Let $batch\_size \in \mathbb{N}_{>0}$ denote the number of samples in each batch. By batch we mean the subset of the data-points which are evaluated before the model's internal parameters are updated. During each 'pass', $\frac{|Y_{train}|}{batch\_size}$ batches are taken from Y_{train}, where $|Y_{train}|$ is the size of set Y_{train}.

Next, we need a regularization to prevent over-fitting to the training data, which would lead to reduced validation accuracy. L_2 regularization is not effective for Adam, as demonstrated by Loshchilov, *et al.* [23]. Since 2017, decoupled weight decay regularization was suggested for this problem [23]. However, in 2020, Wei *et al.* [24], demonstrated the benefits of dropout. Decoupled weight decay forces all weights to be close to 0, but not equal to 0. On the other hand, dropout, as the name suggests, means eliminating units and their connections from the neural network to prevent over-fitting [25]. Let $D()$ be a dropout regularization function, randomly zeroing some of the elements of the input tensor with probability $P_0 \in [0, 1)$ using samples from a Bernoulli distribution. Please note, that each factor will be zeroed out independently on every forward call. [24].

Moving forward, let $Embedding(a, a')$ be a lookup function, with the scope of retrieving the embeddings, where a is the size of the dictionary of embeddings, and a' is the size of each embedding vector. Let $n_f \in \mathbb{N}_{>0}$ be the number of factors. Let $user_f$ contain the user factors, while $users$ denotes the set of users in the batch ($user_f \Leftarrow Embedding(users, n_f)$). Let $user_b$ be the user bias ($user_b \Leftarrow Embedding(users, 1)$). Similarly, let $object_f$ contain the object factors, and $objects$ the objects found in the batch ($object_f \Leftarrow Embedding(objects, n_f)$). Finally, let $object_b$ be the object bias ($object_b \Leftarrow Embedding(objects, 1)$).

The NN solver function calculates the loss $loss_{val}$ using $MSPE(\hat{Y}, Y_{val})$, where $\hat{Y}$ is the prediction defined by $\hat{Y} = \sum(D(user_f) \times D(object_f)) + user_b + object_b$. $\hat{Y}$ is the prediction of Algorithm 1, summarized below:.

Algorithm 1 The NN solver for CF

```
function CF($Y_{train}, Y_{val}, lr, \beta_1, \beta_2, batch\_size, n_f, P_0$ )
    for all $batch \in Y_{train}$ do
        $user_f \Leftarrow Embedding(users, n_f)$
        $object_f \Leftarrow Embedding(objects, n_f)$
        $user_b \Leftarrow Embedding(users, 1)$
        $object_b \Leftarrow Embedding(objects, 1)$
        $\hat{Y} \Leftarrow \sum(D(user_f) \times D(object_f)) + user_b + object_b$
        $loss \Leftarrow MSPE(\hat{Y}, Y_{train}))$
        $backpropagation(loss)$
        $Adam(lr, \beta_1, \beta_2)$
    $loss_{val} \Leftarrow MSPE(\hat{Y}, Y_{val}))$
    return $loss_{val}$
```

C. Identifying the Hyper-Parameters

Let $PN = \{P_1, P_2, \ldots, P_\eta\}$ be the set of all hyper-parameters (the convention used by Wang, *et al.* [14]). In this research we differentiate between two sets of hyper-parameters. Let the automatic optimization candidate set be PO, and let the manually set or predetermined hyper-parameters be PM, so that $PO \cup PM = PN$, and $PO \cap PM = \emptyset$. The PN set's cardinality is $|PN| = \eta$, where η is the total number of hyper-parameters in our method.

In our hybrid HPO approach we aim to minimize $|PO|$, because every additional hyper-parameter in PO leads to a significant increase in time needed to find the optimal parameters. Because $|PO| + |PM| = \eta$, minimizing PO can be achieved by maximizing PM. To maximize PM, we select an optimizer. Currently the best optimizer known to the researchers for this purpose is the Adam optimizer as introduced by Kingma *et al.* [21] (i.e., the one we described above). With this choice, we reach $\eta = 6$, and $PO = \{lr, \beta_1, \beta_2, batch\_size, n_f, P_0\}$.

There is no default good value for $batch\_size$, but we still consider it as a member of the PM set, because we can find the ideal batch size based on the hardware used for training, without the need for automatic HPO. When it comes to determining the batch size for GPU-based training, the first approach is to try larger batch sizes, because that allows faster training due to the parallelism of GPU. However, increasingly larger batch-sizes have a negative effect on the generalization, resulting in worse validation error, if all other hyper-parameters are kept the same. Moreover, an increasing batch size leads to an increasing percentage of time spent on getting training batches ('get_train_batch' step of the implementation). In the next section we will determine this value for the test GPU empirically.

We can conclude that $PO = n_f, P_0$, in other words, we need to decide the number of factors and the dropout probability automatically.

To determine dropout probability we first need to define the search range. Very high dropout values would defeat the purpose of the algorithm and would lead to very slow learning times. On the other hand, setting this value to zero would lead to over-fitting the train data, which is exactly what was intended to be avoided in the first place. On the other hand, we can't be certain that a specific dataset needs dropout to begin with. Therefore, we will trial zero dropout as a possibility. Based on this we suggest the following: $P_0 \in [0, 0.8]$.

To define the search range for the number of factors, $n_f \in \mathbb{N}_{>0}$. While we will consider $n_f = 1$ for the sake of the following experiments, this approach results in an oversimplified linear model, where each user and each object has a single scalar multiplier and a single scalar bias. This model is the least computationally complex to apply, but as a drawback, it leads to a very low recall. A recommender engine based on this linear model would be reluctant to recommend any object, unless there is a very good certainty that it will be purchased, reducing both the number of true positives and false positives. This is not the desired behavior for most use cases. At the time of writing this paper, we have found no scientific method on setting the number of embedding dimensions. Google recommends setting the embedding vector dimension to the fourth root of the number of categories, as a 'general rule of thumb' [26]. We have doubled that number as the upper boundary of the search space, but lower numbers are suggested to save time in real-world scenarios.

We call the training and validation of any given $\{P_0, n_f\}$ value pair with algorithm 1 "a trial". Theoretically, a trial of each possible value of P_0 and n_f (to a certain precision), for many epochs, would result in finding the best PO', but that is obviously not optimal. We need a way to stop trials as soon as we know that they will not produce better results, compared to what we have achieved previously. Therefore, we introduce a pruner, also known as automated early stopping (Liaw *et al.* [27]). A pruner is a method which stops the trial if the best batch result is worse than the median of batch results of previous trials calculated at the same step.

D. Establishing the Sampling Strategy

The last element of our HPO solution is the sampling strategy. There are two valid sampling strategies: independent sampling, and relational sampling. Independent sampling assumes no correlation between hyper-parameters. Conversely, relational sampling, assumes a relationship between the hyper-parameters [28]. We have found no statistical evidence for any relationship between our two hyper-parameters, so our sampler must be based on independent sampling. The recommended Optuna sampler fits this criteria, and it is based on Tree-structured Parzen Estimator by Bergstra *et al.* [29], [30].

IV. Experimental Assessment

A. Implementation Details

We have used Python (version 3.7.7), the PyTorch open source library (version 1.5) and the PyTorch Lightning lightweight wrapper [31] (version 0.7.6) to implement the solution described in the previous section. The Jupyter Notebooks can be found on the project's repository on GitHub: https://github.com/WSzP/uxml-ecommerce

The test machine used in experimental assessment has the following configuration: Intel Core i9-9900K CPU @ 3.60GHz; 64 GiB RAM; NVIDIA GeForce RTX 2080 Ti GPU with 11GiB VRAM.

B. The Dataset

To validate the developed approach documented in this paper, a CF dataset of considerable size was needed, which has no established set of hyper-parameters to influence the search. We created the dataset from real world eCommerce data, published by Kechinov in December 2019. The 'eCommerce Events History in Cosmetics Shop' dataset [32] found on Kaggle contains 8738120 rows and 9 columns. It is worth noting that the method described in this paper is effective regardless of the size of the dataset, but the accuracy will increase with the size.

To convert event data to a sparse matrix we used the data reduction method described by Szabo and Genge [33]. Then, we have applied the three-way data split on the sparse matrix of user-object pairs. Each data-point had a 0.7 chance to be in the train set, a 0.15 chance to be in the validation set, and a 0.15 chance to be in the test set.

C. Evaluation Metrics

The goal of the evaluation metrics for CF is to evaluate how accurately the recommender engine predicts unknown data-points, in other words, how close is the output of the machine learning model to reality. It is worth noting that we have used multiple methods to calculate most metrics to ensure that no calculation method returns significantly different results.

To ease comparison with other related techniques, root means squared error is also given for test results in this paper, but that is simply the square root of the MSPE. For testing purposes, the mean absolute error was calculated using Eq. (2) for q data points. Please note that for comparing MSPE, RMSE, and MAE results, the smaller value is better.

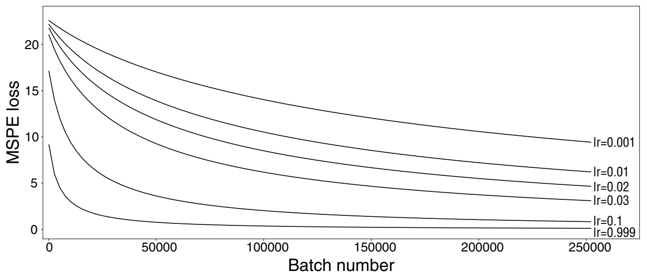

Figure 1. Baseline matrix factorization validation loss.

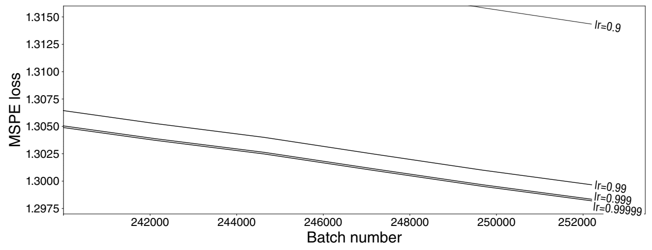

Figure 2. Baseline matrix factorization validation loss (zoomed to the last steps).

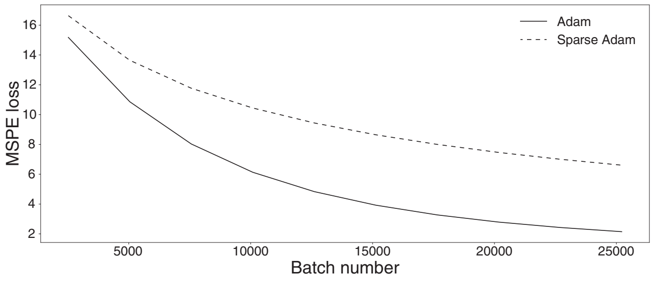

Figure 3. Validation loss (MSPE) differences between SparseAdam and Adam in the first 10 epochs, using the same hyper-parameters.

$$MAE(\hat{Y}) = \frac{1}{q}\sum_{i=n+1}^{n+q} |Y_i - \hat{Y}| \tag{2}$$

The full test dataset precision is calculated by dividing the number of true positives by the number of all positive results. The full test dataset recall is calculated by dividing the number of true positives with the total of true positives and false negatives. Finally, F_1 is the harmonic mean of precision and recall (Eq. (3)). Precision, recall, and F_1 values are considered better if the results are closer to 1.

$$F_1(\hat{Y}, Y) = 2\frac{Precision(\hat{Y}, Y) \times Recall(\hat{Y}, Y)}{Precision(\hat{Y}, Y) + Recall(\hat{Y}, Y)} \tag{3}$$

D. The Baseline Matrix Factorization Training

Dacrema, *et al.* [10], warned the research community in 2019, about a worrying trend in CF research: Among the 18 deep learning CF algorithms 'that were presented at top-level research conferences in the last years', only 7 methods could be reproduced with reasonable effort. 6 of 7 can be outperformed with simple heuristic methods. While the remaining 1 outperformed the baselines, did not outperform a non-neural linear method, at least not consistently.

To serve as a baseline we have built a baseline matrix factorization model, based on established related techniques [34]–[38]. The model uses SGD similar to Bottou's solution [39]. It is an unbiased implementation with no regularization; the simplest form of SGD for the CF problem at hand. As shown in Fig. 1, the validation error improves with the increase of the learning rate, regardless of how many epochs are used to train the model. The difference between the validation loss for 0.9, 0.99, 0.999 and 0.99999 learning rates gradually decreases, but it is still present after 100 epochs, as demonstrated in Fig. 2. The fact that close to 1 learning rate produces the best validation and train loss, suggests that momentum is needed [40]. Overall, the best validation error was produced by learning rate 0.99999, but it had nearly identical results after 100 epochs (i.e., 0.999).

E. Manual Hyper-parameter Optimization

We conducted a manual hyper-parameter optimization experiment and a grid search prior to the hybrid method suggested in this paper.

At the time of writing, only two PyTorch optimizers supported sparse gradients on CUDA: SGD and SparseAdam. SGD was used in the baseline model. SparseAdam is a lazy implementation of Adam because only moments that are present in the gradient are updated, and only those specific parts of the gradient are applied to the parameters.

On the same train and validation matrix, using the same hyper-parameters (batch size: 1024, lr: 0.001, β_1: 0.9, β_2:0.999, n_f: 20, P_0: 0.02), using Adam provided much better results than SparseAdam. This is also shown on the 10 epoch validation error graph in Fig. 3. The only difference between the two implementation variants was that for the SparseAdam variant, we used Sparse Embeddings, instead of dense ones that were used in the case of Adam. This resulted in a much worse final MSPE value (6.6071 for Sparse Adam, 2.1477 for Adam), but also slower training time (247s for Sparse Adam, 154s for Adam). It is worth noting that while back-propagation is faster for Sparse Adam (22.3s vs. 60.3s), the optimizer step is much slower (152.6s vs. 20.5s), while all other steps take roughly the same time. For this reason, we have chosen dense embeddings and the dense Adam implementation. Elsewhere in this paper, when Adam is referred, we mean the dense implementation, instead of the sparse one.

We found that, for Adam, increasing β_1 or β_2 (for example $\beta_1 = 0.99$ and $\beta_2 = 0.99999$) doesn't improve prediction accuracy, and also leads to a slower convergence.

On the other hand, the number of factors (n_f) has a more significant impact on MSPE, but also on full dataset precision and recall. This is summarized in Table 4. To

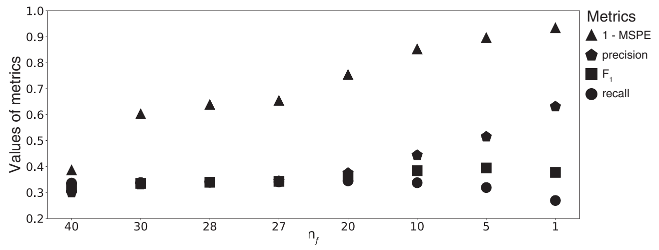

Figure 4. The impact of n_f values on different metrics after training for 100 epochs.

ease the comparison, instead of MSPE, 1-MSPE was used. Therefore, the best possible value for all four metrics listed becomes 1. At $n_f = 28$ the precision becomes equal to the recall, while at higher values, we get higher recall and lower precision. The $1 - MSPE$ and F_1 metrics show a strong positive correlation, with a Pearson coefficient of 0.9521. Please note that using Google's 'general rule of thumb' [26] results in $n_f = 20$ (rounded down to the nearest integer).

While using the same set of other hyper-parameters, the best results are achieved with $n_f = 1$. However, this approach results in an oversimplified linear model, as discussed in the previous section, so the model will calculate how likely a user is to buy anything and how likely it is for an item to be bought.

When training the model for 2030.3 s with a batch size of 128k (131072), the time spent on 'get_train_batch' was of 1998.7 s, while forward and backward steps took a total of 8.6 s and 18.1 s, respectively. Therefore, until a much faster data loading mechanism is found, we do not recommend large batch sizes. For the test GPU, any batch size above 1024 led to a sub-optimal GPU utilization due to the increasing amount of time needed for getting the train batches, but unless the batch size is much higher the effect is minimal. For example, training for 13 minutes leads to near-identical validation errors (0.6476 and 0.6504) for batch sizes of 4096 and 1024. On the other hand, smaller batch-sizes, such as 256, 512, or even 768, lead to worse validation results when trained for the same amount of time. Overall, the majority of the test results given in this paper were made with 1024 batch size, and that is the recommendation of the authors. Please note, that this batch-size is only indicative for GPU training on similar devices to the test device.

To test the relationship between P_0 and n_f the null hypothesis was that n_f and P_0 will have significant positive correlation to each other and significant negative correlation to $MSPE$. Some of the meaningful results of tuning P_0 are summarised as Fig. 5. For the observed dataset F_1 values kept increasing until $P_0 = 0.1$, and only a slight decrease can be seen at 0.2. Contrary to the case with n_f, the correlation of $1 - MSPE$ and P_0 is not as strong (0.6896). For this reason, we assumed $P_0 = 0.2$, which has a F_1 value nearly equal to the maximum (0.385292, the maximum being 0.385763), while also benefiting from excellent MSPE

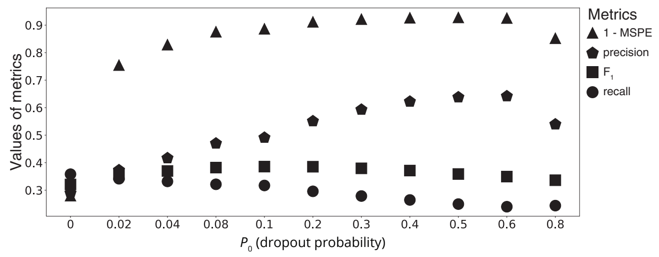

Figure 5. The impact of P_0 values after training for 100 epochs

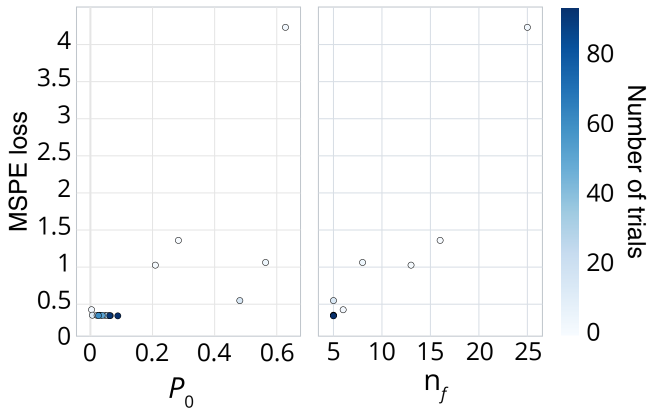

Figure 6. Slice plot showing the number of trials for different P_0 and n_f values, and the resulting MSPE of the trials.

value.

F. Hybrid Hyper-parameter Optimization

The hybrid solution described in this paper produces better hyper-parameters than our manual approach above in 100 trials. After 100 finished trials, the best trial had the MSPE value of 0.3430, and it used the following parameters: $n_f = 5$ and $P_0 = 0.08822302993566566$. The elapsed time was under 2 hours (6685.85 seconds).

To illustrate pruning, we have created a slice plot showing the number of trials with different hyper-parameter values, and the resulting MSPE loss as Fig. 6. To illustrate the effects of the hyper-parameters on the loss function, a contour plot was also created as Fig. 7.

G. Accuracy Assessment

We have trained the same model using the hyper-parameters found by manual optimization and the hybrid method described in this paper, and we have found that the hybrid method's parameters result in a faster convergence and better MSPE values during the same epoch, as shown in Fig. 8. The final results are summarized in Table I.

V. Conclusion

The hybrid HPO solution presented in this paper can be used to find the best hyper-parameters for CF using Adam optimizer or one of the other modern SGD optimizers. One of the main contributions of this paper is the time-efficient

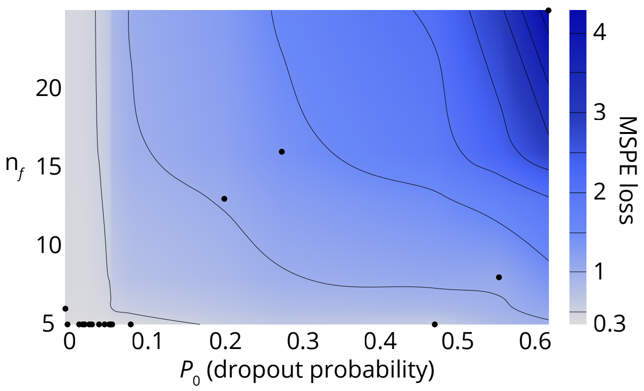

Figure 7. Contour plot of the hybrid HPO, showing the effects of different P_0 and n_f values on MSPE loss.

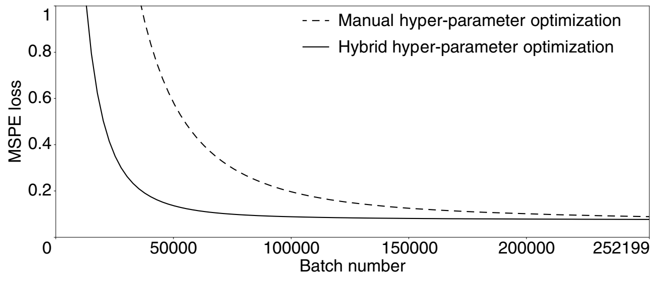

Figure 8. Training for 100 epochs using the hyper-parameters from manual and hybrid HPO.

HPO method, which can find usable hyper-parameters in a relatively short time on a real-world CF dataset. As future work, we intend to find a faster data loading mechanism. Recently a new variant of Adam, AMSGrad, was developed [41]–[43]. The preliminary evaluations suggest slightly worse results for real-world CF datasets. Still, we will analyze other Adam variants tailored to CF, ideally resulting in even better $RMSE$ and F_1 scores.

Table I
EVALUATION OF MANUAL AND HYBRID HPO FOR ALGORITHM 1, COMPARED TO THE BASELINE. (ALL 3 TRAINED FOR 100 EPOCHS EACH.)

	Baseline	**Algorithm 1**	
	Manual HPO	Manual HPO	Hybrid HPO
$RMSE$	1.144820	0.296800	0.276995
MAE	0.589327	0.193500	0.185018
Precision	0.136623	0.552044	0.572273
Recall	0.073204	0.295909	0.305263
F_1	0.095329	0.385292	0.398146

REFERENCES

[1] M. Patil and M. Rao, "Studying the contribution of machine learning and artificial intelligence in the interface design of e-commerce site," in *Smart intelligent computing and applications*. Springer, 2019, pp. 197–206.

[2] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*. Springer, 2007, pp. 291–324.

[3] R. Pan, C. Ge, L. Zhang, W. Zhao, and X. Shao, "A new similarity model based on collaborative filtering for new user cold start recommendation," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 6, pp. 1388–1394, 2020.

[4] V. Vijayakumar, S. Vairavasundaram, R. Logesh, and A. Sivapathi, "Effective knowledge based recommender system for tailored multiple point of interest recommendation," *International Journal of Web Portals (IJWP)*, vol. 11, no. 1, pp. 1–18, 2019.

[5] P. W. Szabo and Z. L. Janosi, "Using machine learning and behavioural analysis for user-tailored viewer experience," in *Proceedings of the 2019 IBC Show*, ser. IBC2019, 2019.

[6] Y. Zhang, Y. Li, Z. Sun, H. Xiong, R. Qin, and C. Li, "Cost-imbalanced hyper parameter learning framework for quality classification," *Journal of Cleaner Production*, vol. 242, p. 118481, 2020.

[7] J. Zobitz, T. Quaife, and N. K. Nichols, "Efficient hyper-parameter determination for regularised linear brdf parameter retrieval," *International Journal of Remote Sensing*, vol. 41, no. 4, pp. 1437–1457, 2020.

[8] A. Shin, D.-J. Shin, S. Cho, D. Y. Kim, E. Jeong, G.-I. Yu, and B.-G. Chun, "Stage-based hyper-parameter optimization for deep learning," *arXiv preprint arXiv:1911.10504*, 2019.

[9] Z. Cai, Y. Long, and L. Shao, "Classification complexity assessment for hyper-parameter optimization," *Pattern Recognition Letters*, vol. 125, pp. 396–403, 2019.

[10] M. F. Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? a worrying analysis of recent neural recommendation approaches," in *Proceedings of the 13th ACM Conference on Recommender Systems*, ser. RecSys '19. Association for Computing Machinery, 2019, p. 101–109.

[11] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 515–524.

[12] L. Zheng, C.-T. Lu, F. Jiang, J. Zhang, and P. S. Yu, "Spectral collaborative filtering," in *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018, pp. 311–319.

[13] Z. Lu, C.-K. Chiang, and F. Sha, "Hyper-parameter tuning under a budget constraint," *arXiv preprint arXiv:1902.00532*, 2019.

[14] C. Wang, H. Wang, C. Zhou, H. Chen, J. Li, and H. Gao, "Experiencethinking: Hyperparameter optimization with budget constraints," *arXiv preprint arXiv:1912.00602*, 2019.

[15] C. Lemaire, A. Achkar, and P.-M. Jodoin, "Structured pruning of neural networks with budget-aware regularization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9108–9116.

[16] O. Goldreich, "Computational complexity: a conceptual perspective," *ACM Sigact News*, vol. 39, no. 3, pp. 35–39, 2008.

[17] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[18] S. Yadav and S. Shukla, "Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification," in *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, Feb 2016, pp. 78–83.

[19] A. K. Nandi and H. Ahmed, *Classification Algorithm Validation*. IEEE, 2019, pp. 307–319. [Online]. Available: https://ieeexplore.ieee.org/document/8958927

[20] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[23] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[24] C. Wei, S. Kakade, and T. Ma, "The implicit and explicit regularization effects of dropout," 2020.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[26] Google, "Introducing tensorflow feature columns," Google Developers Blog, 2017. [Online]. Available: https://bit.ly/tf-fc

[27] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[28] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[29] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[30] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," *Proceedings of Machine Learning Research*, 2013.

[31] W. Falcon *et al.*, "Pytorch lightning," https://github.com/PytorchLightning/pytorch-lightning, 2019.

[32] M. Kechinov, "ecommerce events history in cosmetics shop - november 2019," 12 2019, kaggle dataset, provided by the REES46 Marketing Platform. [Online]. Available: https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop

[33] P. Szabo and B. Genge, "Efficient conversion prediction in E-Commerce applications with unsupervised learning," in *The 28th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2020)*, Hvar, Croatia, Sep. 2020, under review.

[34] H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems." in *IJCAI*, 2017, pp. 3203–3209.

[35] Q.-Y. Hu, Z.-L. Zhao, C.-D. Wang, and J.-H. Lai, "An item orientated recommendation algorithm from the multi-view perspective," *Neurocomputing*, vol. 269, pp. 261–272, 2017.

[36] C.-D. Wang, Z.-H. Deng, J.-H. Lai, and S. Y. Philip, "Serendipitous recommendation in e-commerce using innovator-based collaborative filtering," *IEEE transactions on cybernetics*, vol. 49, no. 7, pp. 2678–2692, 2018.

[37] L. Huang, Z.-L. Zhao, C.-D. Wang, D. Huang, and H.-Y. Chao, "Lscd: Low-rank and sparse cross-domain recommendation," *Neurocomputing*, vol. 366, pp. 86–96, 2019.

[38] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. W. Schuller, "A deep matrix factorization method for learning attribute representations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 3, pp. 417–429, 2016.

[39] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[40] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[41] J. R. Sashank, K. Satyen, and K. Sanjiv, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[42] P. T. Tran *et al.*, "On the convergence proof of amsgrad and a new version," *IEEE Access*, vol. 7, pp. 61 706–61 716, 2019.

[43] T. Tan, S. Yin, K. Liu, and M. Wan, "On the convergence speed of amsgrad and beyond," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 464–470.