# Improving conversion rate prediction via self-supervised pre-training in online advertising

Alex Shtoff Yahoo Research Haifa, Israel alex.shtoff@yahooinc.com Yohay Kaplan Yahoo Research Haifa, Israel yohay@yahooinc.com Ariel Raviv Yahoo Research Haifa, Israel arielr@yahooinc.com

Abstract—The task of predicting conversion rates (CVR) lies at the heart of online advertising systems aiming to optimize bids to meet advertiser performance requirements. Even with the recent rise of deep neural networks, these predictions are often made by factorization machines (FM), especially in commercial settings where inference latency is key. These models are trained using the logistic regression framework on labeled tabular data formed from past user activity that is relevant to the task at hand.

Many advertisers only care about click-attributed conversions, which are conversions that occurred after a user has clicked on an ad. A major challenge in training models that predict conversions-given-clicks comes from data sparsity - clicks are rare, conversions attributed to clicks are even rarer. However, mitigating sparsity by adding conversions that are not clickattributed to the training set impairs model calibration, causing the mean prediction to no longer converge to the actual CVR. Since calibration is critical to achieving advertiser goals, this is infeasible.

In this work we use the well-known idea of self-supervised pre-training, and use an auxiliary auto-encoder model trained on all conversion events, both click-attributed and not, as a feature extractor to enrich the main CVR prediction model. Since the main model does not train on non click-attributed conversions, this does not impair calibration. We adapt the basic self-supervised pre-training idea to our online advertising setup by using a loss function designed for tabular data, facilitating continual learning by ensuring auto-encoder stability, and incorporating a neural network into a large-scale real-time ad auction that ranks tens of thousands of ads, while conforming to the strict latency constraints, and without incurring a major engineering cost. We evaluate our approach and show improvements both offline, during training, and in an online A/B test. Following its success in A/B tests, our solution is now fully deployed to the Yahoo native advertising system, and its impact is measured in millions of dollars annually.

## I. INTRODUCTION

The Yahoo native ads marketplace serves users with native ads that resemble the surrounding content (see Figure 1). It shows billions of daily impressions with a run-rate of several hundreds of millions USD each year. In order to rank native ads for incoming users and their specific context according to the cost-per-click price type, the expected revenue of each ad is computed as a product of the advertiser's bid and the predicted click probability. The click probability is produced by a feature enhanced collaborative filtering algorithm called OFFSET [1].

979-8-3503-2445-7/23/\$31.00 ©2023 IEEE

Clearly Spaces
Clear

Fig. 1. A native ad on Yahoo homepage that resembles the surrounding content.

Advertiser bids, that should reflect the value of an ad click to the advertiser, are either specified manually, or chosen programmatically according to several strategies designed to meet various advertiser goals. At the heart of programmatic bid optimization lie OFFSET models that compute, for each ad, the predicted conversion rate (pCVR) given a click for a given user and a context. For example, one of the strategies allows advertisers to specify the sum they are willing to pay for a conversion, also known as the target cost per acquisition (tCPA). The bid, in that case, is

$$bid = pCVR \cdot tCPA$$
 .

Clearly, calibration of the pCVR model, in the sense that its average prediction approximates the true CVR on any traffic segment, is crucial. Consistent over-prediction causes overbidding, which results in the advertisers paying more than they desire per conversion. Under-prediction causes underbidding, which results in the advertisers winning less auctions and paying less for the ones they won, meaning the advertisers are losing exposure, and we are losing revenue.

OFFSET is a variant of a factorization machine [23] that is trained using the logistic regression framework involving positive and negative events. OFFSET trains incrementally on an infinite stream of data, with periodic checkpoints for hyperparameter tuning [2] and deployment to the production environment. The incremental training methodology was adopted to make sure that the delay between an event happening and a model trained on that event being deployed to production is as small as possible. In many systems [16], [26], [34], including ours, model freshness is of paramount importance for the performance of content recommendation systems deployed in a changing environment.

Training CVR prediction model poses many key challenges, one of which is the data sparsity issue. Accurate models require a large amount of data, but the required amount of data when training conversion models is not always available. For example, typically a few percents of the impressions result in clicks, and a few percents of those result in a conversion. Thus, the number of clicks and conversions for training is quite small, and the coverage of various feature combinations, such as "25-30 years old people using a mobile device", is even smaller.

An initial idea that might come to mind is enriching the training set with additional data, such as conversions that are not attributed to clicks, or impression data in addition to clicks. However, since we require the model for the regression task of CVR prediction, rather than a classification task, it is unclear how the labels of such events should be assigned to produce a calibrated model.

To avoid feeding the main CVR prediction model with data that may impair calibration, we use the celebrated approach of self-supervised pre-training. We pre-train an auto-encoder model [7], [14] on *all* conversions, both related and unrelated to the task at hand. Then, we train the main CVR prediction model *only* on the relevant data while using the encoder's output, which we refer to as the *code*, as an additional feature, meaning that the we attach task-specific prediction layers on top of the encoder. Since these layers are trained only on the task-specific data, no bias is introduced. Moreover, since the code is low-dimensional, training a low dimensional model is potentially immune to data sparsity.

The objective of our design is to provide the CVR prediction model with valuable information about user conversion patterns available in the additional data present in non click-attributed conversions, without accessing it directly. The results of the online and offline experiments in this paper support our thesis by demonstrating improved performance of our CVR prediction model, both offline and in an online A/B test.

The setting of online advertising and incremental training pose unique challenges that require adapting the standard self-supervised pre-training methodology. Hence, we present several design decisions for tailoring the methodology towards a system performing auctions under strict latency constraints, driven by a factorization machine variant that is incrementally trained on tabular data. Concretely, the decisions include the choice of a loss function, careful feature selection, and a model architecture that is driven by a set of criteria and metrics designed to evaluate an incrementally training auto-encoder.

To summarize, the main contributions of our paper are:

- An application of self-supervised pre-training to utilize additional data from conversion events that a CVR model cannot access directly during training.
- A set of techniques for training and using auto-encoders with incrementally trained factorization machine variants that are used in ranking under strict latency constraints.

# II. RELATED WORK

Our work primarily addresses the task of conversion prediction [33]. This task is similar to click prediction, the 'core' task of online ad models, and often uses similar approaches. Its distinct challenges, such as the click-to-conversion delay, the difficulty in attribution to a specific event, and the attribution gap between different platforms and advertisers have been tackled in [10], [17], [18], [27] and many subsequent works.

In this work we adapt a self-supervised pre-training technique to the task of conversion rate prediction in a strict latency constraints settings. Self-supervised pre-training is the technique that leverages self-supervised learning [11], [19] to generate feature representations, typically using a deep neural network, that are (somewhat) agnostic of the overall prediction task and subsequently injecting them in some fashion to the dedicated predictive model. In previous works this main model has also been some type of deep-neural networks model [28], [30], [35]. In [20] a universal user representation is learned from multiple forms of data and then applied to multiple predictive models. [36] considers a sequential ordering of predictive tasks and injects representations learned from multiple tasks into various points of a self-attentive network. [13], [31] suggest ways of taking the standard deep learning predictive flow (i.e., embedding layer into cross layers into predictive output) and injecting intermediate output of one task's model into the other (and vice versa) to boost the performance of both.

In contrast to recent works that tackle this task using deep neural networks, our underlying model is a variant of a factorization machine [23], specifically OFFSET [1], [2]. This is also the underlying model for most prediction tasks in Yahoo's native advertising network.

## III. OUR APPROACH

In this section we describe the techniques we use to integrate an auto-encoder into the CVR prediction models in a way that facilitates incremental training, and enables real-time ad auctions under strict latency constraints.

## A. Training a CVR model with an autoencoder

Like many probability-based regressions models, OFFSET models compute their output (in this case, a pCVR) with the *sigmoid* function:

$$pCVR(\Omega) = \frac{1}{1 + e^{-f(\Omega)}}$$

where  $\Omega$  is the user, ad and contextual properties of the given event. The f used by OFFSET can broken into:

$$f(\Omega) = \langle \mathbf{U}, \mathbf{A} \rangle + \sum_{i \in S} w_i x_i + b \tag{1}$$

where U is a latent representation vector of the user and contextual features, A is a latent representation of the ad features, S is a set of potential additional features,  $x_i$  is an indicator the feature i,  $w_i$  is the learned weight for feature i and b is a global bias. The continual model training process

is responsible for generating the latent representation for all possible combinations of user, contextual and ad features as well as the various  $w_i$ 's and the global bias term. For more on how OFFSET constructs the various latent vectors from  $\Omega$  and its parameters please see [1] and [2]. S is used for features that aren't separable between the user and ad, such as how often and how recently this user has seen this ad, see [3] for more. The training process uses the *logloss* function as its error function:

$$\mathcal{L}(\Omega, y, t) = -(1 - y) \log (1 - pCVR(\Omega)) - y \log pCVR(\Omega)$$

where y is an indicator for a positive event (in our case, click attributed conversion)

We quickly note that this form of computation has benefits for the ability to handle real-time inference. Each ad has it precomputed latent representation, while each user has a singular latent vector (calculated at inference time).

We want to introduce the code generated by the autoencoder into this computation. Since the code depends on both user and ad features, it makes sense to include it in the same way that S is used, so as not to intrude on the portions of the computation that are user/ad isolated. So the resulting f is

$$f(\Omega) = \langle \mathbf{U}, \mathbf{A} \rangle + \sum_{i \in S} w_i x_i + b + \langle \mathbf{W}, C(\Omega) \rangle$$
(2)

where  $C(\Omega)$  is the code associated with a specific event, and **W** is vector of weight learned by the model as part of the training process. The resulting system is illustrated in Figure 2. Learning a linear function of the code is our way to re-use the existing training and serving infrastructure that is built for OFFSET models, that already include a linear term. Although at first glance it might look like having a limited representation power, we show in later sections how we significantly enhance it using a classical method in machine learning.

So a single training interval of the model does the following:

- 1) in interval t load previous OFFSET model  $M_{t-1}$  and previous encoder  $Enc_{t-1}$
- 2) For each event  $\Omega$ , calculate the code  $C = Enc_{t-1}(\Omega)$
- 3) (CVR training) adapt model parameters (including W) using SGD according to  $f(\Omega, C)$  using logloss error
- 4) (Autoencoder trainig) adapt parameters of the autoencoder, whose encoder is  $Enc_t$ .

## B. Architectural challenges

Despite the architecture's apparent simplicity and ease of integration into an existing serving system, it poses several challenges. We describe them here, and present our solution in the following sub-sections. The first challenge stems from incremental training. The basic premise is that the model has long-term memory, and its parameters at interval t - 1 are a good initialization point for training at interval t. Therefore, the codes  $C(\Omega)$  need to be stable, in the sense that the expectation  $\mathbb{E}_{\Omega}[Enc_{t-2}(\Omega) - Enc_{t-1}(\Omega)]$  is small. Otherwise, the premise is broken, since the vector W that OFFSET learned in the previous interval t.

Second, the expressive power of the linear function  $\mathbf{W}^T C(\Omega)$  is weak. Indeed, typically the prediction layers on top of an auto-encoder is, by itself, a neural network of several layers. This is because the separation between positive and negative samples in the encoder's latent space is often non-linear.

The final challenge comes from our need to perform fast real-time auctions. In our system, the time it takes to use a neural network for every item in the auction introduces latency that is orders of magnitude beyond our latency constraints.

#### C. Auto-encoder architecture and design

Suppose our data-set comprises of C categorical columns where column *i* has one of  $n_i$  possible values. The encoder's first layer is an embedding layer - each column has an embedding table of dimension  $n_i \times d$ . Next, the embedding vectors are concatenated to form a vector of length  $C \cdot d$ , which is then passed to a multi-layer perceptron (MLP) network to produce the code. The hidden layers use ReLU activations, while the layer that produces the code uses a tanh activation to produce code that is bounded in the unit box.

Since the encoder's input is composed of categorical features, we treat the decoder as a set of multi-class classifiers with a classifier of  $n_i$  classes for every column. To that end, it consists of an MLP that transforms the code to a vector of size  $\sum_{i=1}^{C} n_i$ , that we treat as C blocks of size  $n_i$  each. The last layer applies a log-softmax activation to each block to produce valid multi-class logits. The reconstruction loss is, naturally, the average of cross-entropy losses applied to each block. Our architecture is illustrated in Figure 3. We denote the loss of auto-encoder M on sample  $\Omega$  as  $\operatorname{RecLoss}(M, \Omega)$ , and the code it produces by  $\operatorname{Code}(M, \Omega)$ . We note that the naïve method of reconstructing a concatenation of one-hot encoded feature indicators using the L2 loss did not result in any improvement of the downstream CVR model.

Just like OFFSET, our auto-encoder is trained in an online manner on data sub-divided into intervals  $D_1, D_2, \ldots$ , training on each event only once. The process produces a sequence of auto-encoder models  $M_1, M_2, \ldots$  On our data, the optimizer that produced the best results was stochastic gradient-descent with momentum [21], [24]. We found the online paradigm to produce satisfactory results, and our use of the paradigm is driven both by the practical success of OFFSET and theoretical results about generalization of online learning [9]. We note that in the online learning paradigm there is no test set separate from the training set, and instead each sample is first evaluated and then trained upon.

The architecture we describe is the result of a process that was driven by empirical evaluation of several metrics that represent its quality in terms of learning data patterns, stability, and generalization. We gradually increased the complexity of the encoder until we found its performance to be satisfactory, while meeting the strict latency constraints. Beyond embedding, code, and hidden layer dimensions, we also tried different ways of combining the column embeddings: addition, product, sum of pairwise component-wise products (à la FM),



Fig. 2. Schematic illustration of our self-supervised pre-training framework. On the left - the auto-encoder that is trained on all conversions, both clickattributed and not. On the right - the CVR prediction model that is trained on clicks and click-attributed conversions, and uses the encoder's code for its training data as an additional feature.

sum of pairwise outer product matrices, and concatenation. The best results according to the metrics described below were obtained with concatenation.

We designed our model with several desired properties in mind. Below, we describe these properties and the metrics we used to measure the model's performance against them.

*a)* Small reconstruction loss: We aim at a reconstruction loss that is much smaller than that of uniformly distributed columns:

$$\operatorname{RecLoss}_{t} \equiv \mathbb{E}_{\Omega \sim D_{t}}[\operatorname{RecLoss}(M_{t}, \Omega)] \ll \frac{1}{C} \sum_{i=1}^{C} \ln(n_{i}).$$

*b) Stability:* The code produced by the encoder of the current interval should be close to the one produced by the encoder of the previous interval. Therefore, the following quantity should be as small as possible:

$$\operatorname{Diff}_{t} = \mathbb{E}_{\Omega \sim D_{t}}[\|\operatorname{Code}(M_{t}, \Omega) - \operatorname{Code}(M_{t-1}, \Omega)\|].$$

c) Interval generalization: Previous interval's autoencoder should generalize well to the samples in the current interval, and the following quantity should be close to 1:

$$\operatorname{Gen}_{t} = \frac{\mathbb{E}_{\Omega \sim D_{t}}[\operatorname{RecLoss}(M_{t}, \Omega)]}{\mathbb{E}_{\Omega \sim D_{t}}[\operatorname{RecLoss}(M_{t-1}, \Omega)]}.$$

d) Learning meaningful patterns: The encoder should reconstruct real data well, and random data badly. The random data-set  $R_t$  is of the same length as  $D_t$ , but with column values chosen uniformly at random from the column's dictionary. The reconstruction loss on  $R_t$  is denoted by  $\operatorname{RecLoss}_t^R \equiv \mathbb{E}_{\Omega \sim R_t}[\operatorname{RecLoss}(M_t, \Omega)]$ . The random loss ratio, defined by

$$\text{RandRatio}_t = \frac{\text{RecLoss}_t}{\text{RecLoss}_t^R},$$

should be close to zero, preferably  $< 10^{-2}$  - the reconstruction of real data should be *orders of magnitude* better than that of random data.

We would like to point out that interval generalization  $\text{Gen}_t$  is tightly related to stability  $\text{Diff}_t$ . In fact, generalization implies stability. Intuitively, if the model from the last interval generalizes well to the current interval's data, then its loss gradients w.r.t the data of the current interval are small, and its parameters will not change significantly. In the evaluation section we show that this phenomenon happens in practice.

We observed that the most profound positive effect on all of the above metrics was obtained from embedding concatenation, rather than other strategies of handling the embedding vectors, and from increasing the embedding dimension. Increasing the number of hidden layers beyond a certain amount had little effect, and so was increasing their dimension. At first the above seems to be in contrast to classical machinelearning, where increasing model complexity results in overfitting and reduces generalization. However, it is known [5], [6], [15] that over-parameterized networks, which are networks that are complex enough to achieve near zero training loss, generalize well. As is apparent in our evaluation section, our auto-encoder's training loss is indeed close to zero. This is despite the fact that it is quite small - we have 7 categorical



Fig. 3. Auto-encoder model architecture. For clarity, illustrated with C = 3 tabular columns. The input is used to choose embeddings, that are concatenated, and fed to a multi-layer perceptron to produce the code. The decoder, in turn, is a multi-layer perceptron that acts as a set of classifiers with cross-entropy losses, that are averaged to obtain the output.

columns, the embedding dimension is 20, the code dimension is 12, and the total number of parameters of our encoder is roughly  $10^5$ .

## D. Nonlinear separability

Having learned an encoder, there is no guarantee that a linear function can differentiate well between positive and negative samples in the encoder's embedding space. Classical machine learning methodology suggests using kernel methods [4], [8], [25] as the go-to technique for creating linear models that represent non-linear functions. However, kernel methods require having the entire training set in advance, and thus do not fit incremental training. As a remedy, we use random Fourier features (RFFs) [22] as an approximation of the Gaussian radial basis function kernel. This technique suits incremental training well, since it dictates a simple formula for

transforming the raw  $Code(\Omega)$  into the feature vector  $C(\Omega)$  in Equation (2) with randomly generated matrix **P** and vector **q**:

$$C(\Omega) = \cos(\mathbf{P}\operatorname{Code}(\Omega) + \mathbf{q}).$$

To verify that RFFs have significantly more representation power, we ran a simple experiment of training a linear CVR model based on the encoder's code, with and without RFFs. We observed that RFFs provide a substantial improvement of 15% in *logloss* and 3.6% in AUC. The model improves as the number of rows of **P** increases, until it reaches a plateau at 200 rows. This means that computing  $C(\Omega)$  requires an additional multiplication by by a  $200 \times 12$  matrix, which does not incur any significant latency cost.

## E. Latency constraints

During an ad auction we rank tens of thousands of ads in a matter of milliseconds. Neural network inference for every ad is prohibitive, because it introduced a significant latency cost far beyond the acceptable range in our system. The above is true even for small encoders of a few tens of thousands of parameters. In theory, if we had only user features in the auto-encoder, we could use the code only *once* per auction. However, an auto-encoder that was trained without features of the ads that were responsible for conversions did not improve the downstream CVR model.

To avoid inference for every ad, we use only high-level ad taxonomy features in our auto-encoder. In our system, we have a two-level taxonomy hierarchy - the first level breaks down ads by high-level categories, such as *Finance* or *Tourism*, and the second level by a few dozen categories. Hence, during ranking, we need to compute the product  $W^T C(\Omega)$  from Equation (2) only once for every low-level category, and add the result to the score  $f(\Omega)$  we compute for every ad according to its taxonomy.

#### IV. EVALUATION

We performed several experiments on our CVR models that predict the probability of a conversion given a click. For training our CVR prediction model, we treat clicks as negative events, and conversions attributed to clicks on our properties as positive events. In addition, we also have conversions that are attributed to ad views. All models train on daily intervals.

In this section we first corroborate our thesis that directly adding view-attributed conversions to the training set produces a less accurate model. Then, we evaluate our auto-encoder according to the metrics in Section III-C. Next, we show that equipping a CVR model with an auto-encoder improves the model's accuracy during training. Then, we verify that the incurred computational cost has no visible effect on the auction latency, and demonstrate the improved performance of the model on real traffic via an A/B test against the production model.



Fig. 4. Reconstruction losses over real and random data, of an auto-encoder trained on a month worth of data.

#### A. Training on view-attributed conversions

The paper's premise is that training on conversions that are not attributed to clicks, in addition to those that are attributed to clicks, degrades model accuracy. We trained a model that includes view-attributed conversions as positive events, in addition to the relevant events. In both models, the *logloss* was evaluated only for clicks and conversions attributed to clicks. The *logloss* achieved by the model that includes viewattributed conversion is between 3% and 7% higher (depending on the day) than that of the model which does not include them, which makes such a model much *worse*.

#### B. Auto-encoder evaluation

Here we evaluated the desired properties of our autoencoder using the metrics  $\operatorname{RecLoss}_t^R$ ,  $\operatorname{Gen}_t$ , and  $\operatorname{Diff}_t$  from Section III-C.

In order to validate that our auto-encoder learns meaningful patterns, rather than just the identity function, we measure  $\operatorname{RecLoss}_{t}^{R}$  and  $\operatorname{RecLoss}_{t}^{R}$  for an auto-encoder trained on a month worth of data. We plot the results in Figure 4. It is evident that as the model trains on more data, its performance on true data improves, on random data degrades, the reconstruction loss approaches zero, and the RandRatio<sub>t</sub> approaches  $10^{-4}$ , as desired.

In order to verify that our auto-encoder is stable and generalizes well between consecutive days, we measure  $\text{Gen}_t$  and  $\text{Diff}_t$  over a few weeks worth of data. The results are plotted in Figure 5, where each point is the result of training on a day of data. As is apparent, most of the values of  $\text{Gen}_t$  are close to 1, which means that our auto-encoder often generalizes well. Moreover, as we claimed in Section III-C, generalization and stability are tightly coupled. Indeed, most of the points lie along a straight regression line, and there are no points where  $\text{Gen}_t$  is small but  $\text{Diff}_t$  is large.

# C. CVR model offline lifts

We train a new CVR model using our auto-encoder based approach, and measure the *logloss* difference between the new model and the production model at the time. In contrast



Fig. 5. Daily generalization metric  $Gen_t$  (closer to 1 is better) versus daily stability metric  $Diff_t$  (lower is better).

With vs. without autoencoder



Fig. 6. Daily *logloss* difference (negative is better), in percents, between a model enriched by an auto-encoder, and the production model at the time. Marker size is proportional to the evaluation set size.

to a CVR model that is directly trained on view-attributed conversions, the auto-encoder based model has an improved *logloss*, as is apparent from Figure 6.

#### D. Incurred latency tests

We deployed the new workflow to our production system by periodically loading the encoder that is trained with the model, and using it in ad auctions according to the formula in Equation (2). Since we use only high level ad taxonomy features, our ad auction computes the inner product  $\langle \mathbf{W}, C(\Omega) \rangle$  only once for each ad category, and uses the cached results for all the ads participating in the auction. We compare the latency incurred by auctions where our new model is used to auctions with the production model at that time in Figure 7. Indeed, the size of the model and the usage of high level features paid off - there is essentially no visible latency difference between the new and the product model at the time.

#### E. Online A/B tests

*a)* Setup: A CVR model performance is measured along two axes - the revenue measured by the *average cost per thousand impressions* (CPM), and the percentage of advertiser



Fig. 7. Average hourly latency difference, in milliseconds, between the new encoder-augmented model and the production model at the time. Negative values are better.

spend belonging to campaigns that attain advertiser cost per acquisition (CPA) goals. The objective is obtaining improvement in at least one metric without impairing the other. This is since a better model can either correct under-prediction, which results in lower bids and lower revenue, or over-prediction, which results in missing the advertiser CPA goal. To evaluate the online performance, we launched an online bucket serving 50% of Yahoo native traffic and measured lifts of both metrics against the production model. The evaluation was conducted over a week and included billions of impressions.

b) Results: On average, the online test bucket showed 0.6% CPM lift and 0.67% revenue lift overall. Such a CPM (or revenue) lift translates to many millions of USD in yearly revenue once the solution is deployed to all traffic. The results also indicate a +10.1% increase to the spend that is under oCPC influence, with a slightly larger percentage of spend belonging to campaigns hitting the CPA goal (83.45% vs 82.96% in test and control bucket receptively). In short, the results show that the new model increases the overall revenue while maintaining the advertiser's CPA goals. Following these positive results, the new model was fully deployed and serves 100% of the Yahoo Gemini native traffic.

## V. DISCUSSION AND FUTURE WORK

We described a framework for reducing the effect of data sparsity in CVR prediction by incorporating conversions that are considered un-labaled via self-supervised pre-training. Moreover, we adapted the basic idea to systems based on incrementally-trained factorization machines that are used in ad auctions with strict latency constraints. We have shown that even a simple auto-encoder with a small number of parameters already provides a significant improvement. Therefore, natural future directions, beyond adding more features, should aim to squeeze more information out of the un-labaled conversions by further improvements of model architecture and training techniques.

One direction is looking for better ways to incorporate the encoder's output into the factorization model without incurring a significant latency increase. Currently, we are using the random Fourier features technique as a 'projector network'. It can be viewed as a shallow one-layer network whose activation is the cosine function, and its weights are random, rather than learned. However, the projector does not have to be a random one-layer network. It can be an arbitrary deep neural network with arbitrary activation functions and learnable weights. As long as this network is small enough to facilitate fast inference, it can potentially squeeze out more performance out of the encoder network. Moreover, the projector's incorporation into the factorization model does not have to be in the form of an additive term, but can be in the form of another latent vector. For example, instead modifying the score in Equation (1) as in (2), it can be alternatively modified as

$$f(\Omega) = \langle \mathbf{U}, \mathbf{A}, P(\operatorname{Code}(\Omega)) \rangle + \sum_{i \in S} w_i x_i + b_i$$

where  $\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle = \sum_{i=1}^{n} x_i y_i z_i$  denotes a "triple" inner product, and  $P(\cdot)$  is the projector network whose output dimension is equal to the dimension of the user and ad vectors.

Another prominent direction is using more advanced selfsupervised learning techniques, such as masked auto-encoders [12], or even drifting away from the auto-encoder framework and use techniques such as Barlow Twins [32]. Another improvement can potentially come from better model architectures, such as using small transformers [29] to process feature embedding vectors.

## REFERENCES

- [1] Michal Aharon, Natalie Aizenberg, Edward Bortnikov, Ronny Lempel, Roi Adadi, Tomer Benyamini, Liron Levin, Ran Roth, and Ohad Serfaty. Off-set: one-pass factorization of feature sets for online recommendation in persistent cold start settings. In *Proc. RecSys'2013*, pages 375–378, 2013.
- [2] Michal Aharon, Amit Kagian, and Oren Somekh. Adaptive online hyper-parameters tuning for ad event-prediction models. In *Proceedings* of the 26th International Conference on World Wide Web Companion, pages 672–679. International World Wide Web Conferences Steering Committee, 2017.
- [3] Michal Aharon, Yohay Kaplan, Rina Levy, Oren Somekh, Ayelet Blanc, Neetai Eshel, Avi Shahar, Assaf Singer, and Alex Zlotnik. Soft frequency capping for improved ad click prediction in yahoo gemini native. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM '19, page 2793–2801, New York, NY, USA, 2019. Association for Computing Machinery.
- [4] A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821– 837, 1964.
- [5] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [6] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings* of Machine Learning Research, pages 254–263. PMLR, 10–15 Jul 2018.
- [7] Dana H. Ballard. Modular learning in neural networks. In Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1, AAAI'87, page 279–284. AAAI Press, 1987.
- [8] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

- [9] Nicolò Cesa-bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.
- [10] Olivier Chapelle. Modeling delayed feedback in display advertising. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1097–1105, 2014.
- [11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
- [12] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.
- [13] Guangneng Hu, Yu Zhang, and Qiang Yang. Conet: Collaborative cross networks for cross-domain recommendation. In *Proceedings of* the 27th ACM international conference on information and knowledge management, pages 667–676, 2018.
- [14] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. AIChE journal, 37(2):233–243, 1991.
- [15] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014.
- [16] Andreas Lommatzsch, Benjamin Kille, and Sahin Albayrak. Incorporating context and trends in news recommender systems. In *Proceedings* of the international conference on web intelligence, pages 1062–1068, 2017.
- [17] Quan Lu, Shengjun Pan, Liang Wang, Junwei Pan, Fengdan Wan, and Hongxia Yang. A practical framework of conversion rate prediction for online display advertising. In *Proceedings of the ADKDD'17*, pages 1–9. 2017.
- [18] Mohammad Mahdian and Kerem Tomak. Pay-per-action model for online advertising. In *Proceedings of the 1st international workshop* on Data mining and audience intelligence for advertising, pages 1–6, 2007.
- [19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.
- [20] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. Perceive your users in depth: Learning universal user representations from multiple e-commerce tasks. In *Proceedings of the* 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 596–605, 2018.
- [21] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.
- [22] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.
- [23] Steffen Rendle. Factorization machines. In 2010 IEEE International Conference on Data Mining, pages 995–1000. IEEE, 2010.
- [24] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.
- [25] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.
- [26] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28:2503–2511, 2015.
- [27] Xuhui Shao and Lexin Li. Data-driven multi-touch attribution models. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 258–264, 2011.
- [28] 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, pages 1441–1450, 2019.

- [29] 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.
- [30] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. Parameter-efficient transfer from sequential behaviors for user modeling and recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 1469–1478, 2020.
- [31] Feng Yuan, Lina Yao, and Boualem Benatallah. Darec: Deep domain adaptation for cross-domain recommendation via transferring rating patterns. arXiv preprint arXiv:1905.10760, 2019.
- [32] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings* of Machine Learning Research, pages 12310–12320. PMLR, 18–24 Jul 2021.
- [33] Weinan Zhang, Shuai Yuan, and Jun Wang. Optimal real-time bidding for display advertising. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1077–1086, 2014.
- [34] Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He, Meng Wang, Yan Li, and Yongdong Zhang. How to retrain recommender system? a sequential meta-learning method. In *Proceedings of the 43rd International* ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1479–1488, 2020.
- [35] Yujing Zhang, Zhangming Chan, Shuhao Xu, Weijie Bian, Shuguang Han, Hongbo Deng, and Bo Zheng. Keep: An industrial pre-training framework for online recommendation via knowledge extraction and plugging. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3684–3693, 2022.
- [36] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Selfsupervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 1893–1902, 2020.