Nudging Neural Click Prediction Models to Pay Attention to Position

Efi Karra Taniskidou[*]
efi@amazon.co.uk
Amazon
Edinburgh, Scotland, United Kingdom

Wenjie Zhao[*]
zhawenj@amazon.co.uk
Amazon
Edinburgh, Scotland, United Kingdom

Iain Murray[†]
iamurray@amazon.co.uk
Amazon
Edinburgh, Scotland, United Kingdom

Roberto Pellegrini
peller@amazon.co.uk
Amazon
Edinburgh, Scotland, United Kingdom

ABSTRACT

Predicting the click-through rate (CTR) of an item is a fundamental task in online advertising and recommender systems. CTR prediction models are typically trained on user click data from traffic logs. However, users are more likely to interact with items that were shown prominently on a website. CTR models often overestimate the value of such items and show them more often, at the expense of items of higher quality that were previously shown at less prominent positions. This self-reinforcing *position bias* effect reduces both the immediate and long-term quality of recommendations for users. In this paper, we revisit position bias in a family of state-of-the-art neural models for CTR prediction, and use synthetic data to demonstrate the difficulty of controlling for position. We propose an approach that encourages neural networks to use position (or other confounding variables) as much as possible to explain the training data, and a metric that can directly measure bias. Experiments on two real-world datasets demonstrate the effectiveness of our approach in correcting for position-like features in 2 state-of-the-art CTR prediction models.

CCS CONCEPTS

• **Information systems → Online advertising**; **Retrieval models and ranking**.

KEYWORDS

Click Models; Position Bias; Neural Ranking Models.

ACM Reference Format:
Efi Karra Taniskidou, Wenjie Zhao, Iain Murray, and Roberto Pellegrini. 2023. Nudging Neural Click Prediction Models to Pay Attention to Position. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614994

[*]The first two authors have equal contribution.
[†]Also with the University of Edinburgh. This paper describes work performed at Amazon.

1 INTRODUCTION

Click-through-rate (CTR) prediction is the task to estimate the probability of an item being clicked when it is presented to a user. It is a key task for content selection and ranking in recommender systems. Neural or "deep learning" approaches to CTR prediction are successful as a flexible framework for combining multiple sources of context and information (e.g., [5, 9, 26]). The products shown to users on a retail website are often curated for a particular user, but are in the context of a given widget, at a particular position, on a particular page, at a particular time. Any piece of information relevant to the current ranking task can be potentially represented as a vector of numbers, an *embedding*, and included as input to a neural architecture that chooses which items to show. The embedding representations, and how to combine them, can all be learned as part of the end-to-end training of a single *large* system (e.g., [6]).

Although deep learning methods have been amazingly successful at fitting models with far more parameters than data points, care is still required. A system may perform well on the training data, and held-out validation data generated in the same way. However, models can't generally extrapolate well to all new situations [21]. And if we use a model to evaluate the CTR of all possible items that could be shown, then by design we are evaluating the model at new hypothetical situations.

We first consider dealing with data where past recommendations were shown at different page positions. An item that would be good to show might previously have performed poorly because it was shown at low page positions. If we wrongly attribute the poor outcomes to the quality or relevance of the item itself, we might incorrectly and unfairly choose to not show it. This *position bias* effect reinforces previous show decisions, which themselves could have resulted from poorly-fitted models, or models fitted to outdated data, or data biased by other systems on the website [3, 16].

Because page position is an observed quantity in logged ads data, we can attempt to control for it: we include it as an input to our model to help explain the observed outcomes. However, flexible neural models are often not *identifiable* [22]: a standard training loss alone is not enough to specify how the model should use position to explain the training data. Some models might be formally identifiable given infinite data, but not in practice given noisy finite data. In these cases we require an *inductive bias*, or *regularizer*, that encourages the model to avoid reinforcing position bias.

We demonstrate the difficulty with controlling for variables in neural models using an extreme synthetic setup where we know the ground truth (Section 3). We propose an extra loss that encourages the model to use position as much as possible, potentially at the expense of over-exploring items in future. We then describe how to detect and measure bias in real-world data, before applying our proposed method to existing CTR prediction models (Sections 5). Given how easily the method can be applied in principle to any model architecture, and its effectiveness in removing position bias in our examples, we think it will be useful in many recommendation models. That said, the position bias effect that we describe is only one aspect of building fair recommenders, and makes certain assumptions. We discuss connections to some of the other work on position or popularity bias, and outstanding issues in Section 2.

2 RELATION WITH PREVIOUS WORK

2.1 CTR prediction models

CTR prediction is a fundamental task in a number of industrial applications like advertisement, ranking and recommendation systems. Given its prominence a number of click prediction models have been proposed in the literature. The common theme is that the features used in these models need to interact in a complex non-linear way to effectively explain the observed CTRs.

The Wide and Deep architecture [5] was inspired by the observation that generalised linear models perform relatively well on click predictions and the authors combined a deep neural model with a factorisation machine. However, manual feature engineering is required to model linear and pairwise feature interactions in the wide part of the network.

To mitigate the requirement of manual feature engineering a new architecture called Deep Factorization Machine [9] was proposed. In this architecture the embeddings for input features are shared between the deep and the wide components.

More recently a neural architecture that does not require a wide and shallow component, MaskNet [25] was proposed. In this work the authors propose to use feature masking to encourage a deep multi-layer perceptron to model multiplicative feature interactions.

Our approach to control for positional bias is generic and in principle can be applied to any architecture. Our experiments use two click prediction SOTA models: [9, 25], and show that we can reduce position bias without impacting the overall model performance.

2.2 Positional bias

The first in-depth study of positional bias was performed by Joachims et al [11]. The authors compared explicit feedback with user click data and found that the click data is affected by positional bias and that it can effect models trained to estimate the relevance of an item.

One common debiasing approach is to collect a small unbiased dataset under a uniformly random recommendation policy [4]. Showing items at random intercepts the feedback loop and gathers unbiased user behavior. However, it can also hurt customer experience and platform revenue.

When truly random data is not easily available, Inverse Propensity Weighting (see for example [23]) (IPW), also known as Inverse Propensity Scoring (IPS), is an inexpensive way to create pseudo-random data.

To understand the simplest version of IPW, we can consider a simple item recommendation task, where every item appears equally often at every position on a page. When an item appears at worse, lower down positions, the item will be clicked less. But because every item appears in each position equally often, position effects do not bias the system to prefer any particular item. However, our training data is usually biased, such that the probability of item i occurring at position k, $P(k \mid i)$ is not a constant. Weighting the training loss for each example by the Inverse Propensity, $1/P(k \mid i)$, gives an unbiased "importance sampling" estimate of the loss for the uniformly-sampled positions case.

The weights are only defined if the propensities are non-zero, and IPW can have high variance when the probability of placing some items in some positions is small. The method is also only unbiased if we know the true propensities. There is a rich line of work in different methods for estimating propensities and reducing variance, while trying to reduce the need to show bad items in prominent positions when gathering data [e.g. 1, 8, 12, 20].

An alternative way to use logged positions is to include them in the model of clicks that we fit. A single model f can estimate the probability of click event "$y=1$" for an item i at a given position k:

$$P(y=1 \mid i, k) = f(i, k). \tag{1}$$

For particular situations, previous work has put structure into the click model, reflecting the known layout of the page, or observed user-behavior [e.g. 10, 14, 27, 29].

When we model a conditional click probability (1), we do not need to know the probabilities of showing the items at each position, and there is no fundamental need to weight the data based on some items being shown more prominently than others. When scoring which item to show, we can evaluate the click probability for all items conditioned on the same page position, either the top position, or the position we will place the item in. This flexibility allows us to include other page information in the position features, potentially specializing what items we show depending on the context.

Even if we condition on the observed position, the training data can potentially be biased by unobserved confounding factors. IPW methods also usually make a no-unobserved-confounder assumption [12] and can be biased. Dai et al. [7] use a causal graph to model an unobserved confounder, and use backdoor adjustment to correct for it.

This paper builds on our current understanding of position bias by identifying that flexible neural models can easily suffer from position bias in practice, even with no hidden confounders, and even with models that are asymptotically consistent. The next section illustrates this claim with a simple example, which also motivates our approach. In Section 4.1 we propose a new task to control for position bias and compare our approach with one proposed by [27].

3 SYNTHETIC POSITION-BIAS DEMONSTRATION

We generate synthetic data from a simple model that's sufficient to illustrate a difficulty of controlling for variables when modelling recommendation outcomes with neural networks. Each generated event is a triple (i, k, y), an integer item identifier $i \in \{0, \ldots, 999\}$,

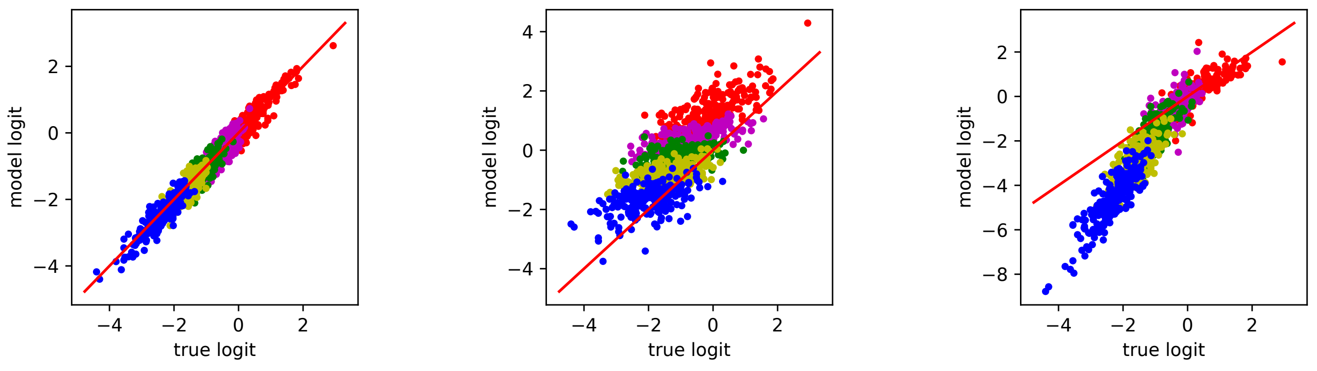


Figure 1: Synthetic feedback experiments. Logit probabilities of a positive outcome under fitted models against ground truth. Each point is for an item shown at position $k = 0$, colored by the training-time position (set by a previous model fit). a) Logistic regression obtains a well-calibrated fit. b) Not controlling for position gives a worse fit. The horizontal color bands indicate a strong position-bias: for each true logit value, the estimated logits are ordered by training-time position. c) A neural network conditioned on position still has some position bias.

a page position $k \in \{0, 1, 2, 3, 4\}$, and a binary outcome: either the user clicked on the recommendation ($y=1$) or not ($y=0$). Each item has a known embedding vector $\boldsymbol{x}^{(i)}$, which we can imagine came from a separate pre-training task.

The binary outcomes are generated from a simple logistic regression model:

$$P(y=1 \mid i, k) = \sigma\big(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b^{(k)}\big), \quad \text{where } \sigma(a) = \frac{1}{1+e^{-a}}, \tag{2}$$

and $b^{(k)}$ are position-specific bias parameters, where $b^{(k)} = -k - 1$. Positive outcomes are most probable in the top page position $k=0$, and for items in the region of embedding-space aligned with $\boldsymbol{w}$.

We initially sampled the page positions uniformly. After fitting a model, we use it to rank the items and simulate a new dataset, creating a deliberately-extreme feedback effect. Each time we show an item from the best fifth of the items (according to the model), we deterministically show the item in position $k=0$. Items in the next fifth of the model's ranking are shown in position $k=1$, and so on.

Other simulation details: We set the true weights $\boldsymbol{w}$ to a vector of ones, and sampled the $(D=50)$-dimensional item embeddings $\boldsymbol{x}^{(i)}$ from a zero-mean independent normal distribution with variance $1/D$, such that the item's contribution to the logit, $\boldsymbol{w}^\top \boldsymbol{x}^{(i)}$, would have variance 1. We sampled 10^4 training examples, with items sampled uniformly at random from a set of 1,000 items. Modelling the long-tail nature of item impression frequencies did not turn out to be necessary for this demonstration.

3.1 Model identifiability and flexibility

If we assume we know the model in (2), then the model is identifiable and well-behaved. As long as there are positive and negative examples at each page position and for enough different items, there is a unique solution, and in the limit of many examples, the ground truth parameters will be recovered.

Estimating the correct value of the items and page positions requires fitting them jointly (Figures 1a–b). In our feedback simulation, items aligned with an estimate of the weights $\boldsymbol{w}$ get placed at better page positions. As we move through item-embedding space in the direction $\boldsymbol{w}$, the click-through rate for the associated events increases — both because the items are more desirable, and because the page position improves. If we ignored the page position of the events, our model will have an *omitted-variable bias* where we attribute all of the improvement to the items themselves, and learn a weight vector that is too large. If our initial estimate of $\boldsymbol{w}$ was incorrect, the next dataset can reinforce that mistake: the page position choices boost the probability of positive outcomes in the estimated direction, so we must control for page position. Similarly, if we fitted a model conditioned on just the page positions, we'd overestimate the usefulness of showing in the best page positions.

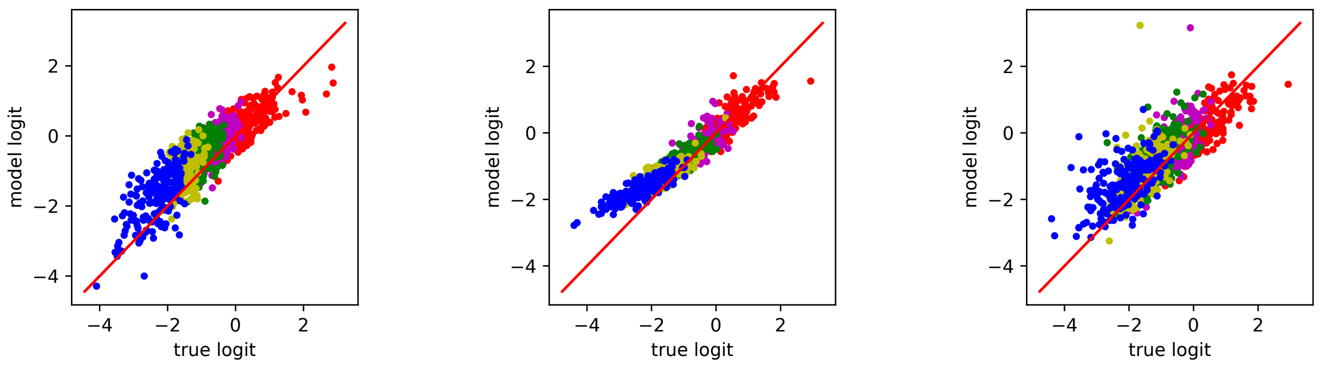
In real recommender systems we can often justify non-linear fits, but at some point the model will become unidentifiable. To illustrate, our synthetic setup only has 1,000×5 = 5,000 item-position combinations, so in principle we could estimate an arbitrary table of outcome probabilities based on counts. However, for deterministic page placement this completely-general model never gets any data for some item-position combinations, and so is not identifiable. Moreover, in our limited-data simulation, some items are never impressed, and some that are impressed never convert. Using models that can generalize between similar items is important.

As a middle-ground we introduce a neural network function f into the model we fit,

$$P(y=1 \mid i, k) = \sigma\big(f(\boldsymbol{x}^{(i)}) + b^{(k)}\big), \tag{3}$$

while still simulating data from a linear model. If f is an arbitrary function, each item can have an arbitrary contribution to the logit and the model isn't identifiable given deterministic placement of items. However, for a well-behaved f, similar items will have similar predictions, and we might hope to pull apart the contribution of the items and the positions. In our simulations, f was a stack of three residual layers with leaky ReLU activations, followed by a linear combination, reflecting neural network layers commonly used in production neural recommenders.

Fitting the neural network to data with page positions chosen uniformly at random works only slightly less well than the ground-truth linear model. However, as soon as the page positions are based on a model fit, a page position bias becomes obvious. At a true logit value of ≈−1 in Figure 1c, the neural network's estimated logits



(a) Neural net. + weight L2 **(b) Neural net. + logit L2** **(c) Neural net. + extra task**

Figure 2: Alternative neural network regularizations in synthetic feedback experiments (as in Figure 1). These are just illustrations of what can happen; the results are sensitive to details such as parameterization and learning rate schedules. a) L2 regularization on weights improves the fit and removes position bias (no horizontal color bands; vertical banding simply reflects successfully identifying best and worst items). b) L2 regularization of logit has a visibly different result, with lower variance estimates of the items' effects. The effect of lower page positions estimated less well, affecting the estimates of items shown at those positions. c) Initializing the model with a position-only model also reduces position bias, in this case with higher variance estimates.

are roughly ordered by the page position used to generate the data. The model is too flexible to know how to attribute the observed outcomes to the positions and the items. In this case we should just use a linear model, but in cases where a linear model underfits, we need to be able to control the fits produced by the neural network.

3.2 Generic regularization methods

We employed *early stopping* to choose a model that appeared to have the lowest validation score. Nevertheless, there is still some overfitting: at the early-stopped model the training score is better than that obtained by a linear model, and the validation score worse.

We can consider other forms of generic regularization, not specific to position-bias effects, such as adding an L2 weight penalty. In our synthetic example, L2 regularization immediately helps (Figure 2a). Unfortunately the details depend on the precise setup of the experiment. For example, if the scale of the item embeddings are changed, then the relative regularization of the item and position features changes. Whether we choose explicitly or not, we need to set which weights to regularize and by how much. In this example, where the ground truth is linear, the correct thing to do is turn off the neural network weights in the residual layers. However, in real models we often can justify non-linear fits, and large L2 regularization may hurt performance in production systems.

Another form of regularization encourages the logit, the final scalar inside the sigmoid output of the classifier, to be small. The weights are now only regularized implicitly, by backpropagating an extra loss applied to the logit. In label-smoothing [24], the logit is penalized according to its KL-divergence with a uniform or reference distribution. More recent work analyzed penalizing the square of the logit [19] and showed (under strong assumptions) that this regularizer encourages the model to use all of its inputs, rather than ignore some of them. This regularizer also improves the fit in our example simulations, although with different trade-offs to an L2 penalty applied to the weights (Figure 2b).

3.3 Regularizing with an additional task

Different regularization methods create different *inductive biases*, that favor what sort of model we will fit. As we know that page position has a strong effect, and that we don't want to reinforce biases from the past, we could attempt to encourage the model to use page position more strongly. However, the above generic regularizations do not explicitly favor features with particular meanings.

As an alternative, we try to use page position to explain the data as much as possible, and only use item when we see we really have to. We first fit a model that ignores the item, and models position with bias parameters c:

$$P(y\!=\!1 \mid k) = \sigma\big(c^{(k)}\big). \tag{4}$$

Then clamping c, we fit the original model's f and b as a correction to this position-only model's logit:

$$P(y\!=\!1 \mid i, k) = \sigma\big(f(\boldsymbol{x}^{(i)}) + b^{(k)} + c^{(k)}\big). \tag{5}$$

We can't omit the original position bias parameters b, because the c parameters do *not* estimate the true causal effects of the positions: they are biased by position bias in the training set, and need correcting. Equation (5) is the same model as in (3), we've just changed the initialization. Given the strong effect of page position, the probabilities start at roughly the correct size, and fitting then jointly introduces item effects and corrects the page positions. Early stopping terminates fitting when the fit starts to become unjustified (no longer generalizes better). Figure 2c shows an example result of this initialization with no explicit regularization in the loss. The obvious position bias has gone, although (in this example) there is more variability in the estimates of the items' underlying relevance than with other forms of regularization. Using this model to recommend items will explore the catalog of possible items more.

Initializing with a position-only model can be combined with other forms of regularization. The inductive bias towards the initialization (created by fitting dynamics and early stopping) could be made explicit, for example by adding a penalty $\lambda \| f(\boldsymbol{x}^{(i)}) + b^{(k)} \|^2$. That is, regularizing the logit towards the position-only model rather than zero as proposed by [19]. However, just as there isn't a generally-dominant regularization method for neural networks in general, we don't have a general recommendation: the best choices depend on the details of the data and models.

4 PROPOSED METHODS

Section 3.3 proposed a regularization method that uses position features by themselves to explain the data as much as possible (4), and then re-fit a model on the "residuals" (5). It assumed no interactions between the position features k and the other features x, which might not hold in the real world. For example in online advertising, we want to consider websites as a positional feature because some websites can have particularly high click through rate. At the same time, we also want to select ads that are related to the topic of the website, which means excluding websites from f is inappropriate.

In this case, we want a method that follows the same underlying idea of using the positional features as much as possible, but allows for more general interactions.

4.1 Extra task

As in Section 3.3 we fit a model that can only use the positional features (Equation 4). If there are multiple position features, we embed them and combine them with a neural network function:

$$P(y=1 \mid k) = \sigma\big(g(k)\big). \tag{6}$$

Then clamping this position-only model, we fit the original model as a correction. To allow generic interactions we modify (5) by allowing the model to be a generic function of all the features

$$P(y=1 \mid x, k) = \sigma\big(f(\boldsymbol{x}, k) + g(k)\big). \tag{7}$$

In practice, for convenience, we fit the sum of the losses for both models together in one training loop, where the position-only model converges quickly. We *detach* the backpropagation of gradients from the main model's loss to the g parameters, so that they are fitted only to the position-only task.

Our approach is similar but not equivalent to Zhao et al. [27]. In the paper the authors add a shallow tower that can only use the position features

$$P(y=1 \mid x, k) = \sigma\big(f(\boldsymbol{x}) + g(k)\big). \tag{8}$$

However, in their approach the position features are omitted from the main model f and there is a single loss. The latter approach does not allow for interactions between the position features k and the other features x, which might or might not be realistic depending on the details of the system generating the data. We call this approach "shallow tower" and we compare it to our proposed extra task in the experiment Section 5.

4.2 Research Questions

The main focus of our paper is to shed light on the following research questions:

(1) Can we detect and quantify positional bias in existing publicly available datasets?
(2) Does the task proposed in Section 4.1 help mitigate any existing positional bias?
(3) How does the proposed task perform when we inject more bias into the training set?

4.3 Quantitative calibration-based bias metric

In order to answer the research questions listed above we need a metric that, given the model's output as a set of triplets (P_i, s_i, y_i) of predicted score $s_i \in [0,1]$ and actual labels $y_i \in \{0,1\}$ for product P_i, can detect whether the model is overestimating or underestimating the value of a particular set of products $S = \{P_1, P_2, \ldots, P_N\}$. The simplest possible solution would be to compare the sum of the scores and the sum of the true labels

$$B(S) = \frac{\sum_i \mathcal{I}(P_i \in S) s_i}{\sum_i \mathcal{I}(P_i \in S) y_i}, \tag{9}$$

where the indicator function $\mathcal{I}$ limits the sum to products in the set we are interested in. The metric is one if the value of the products in S is estimated correctly, above one if the model overestimates the value of the set, and below one if the model underestimates.

The metric $B(S)$ is unsuitable for the click prediction task because the typical click through rate is relatively low and the sums are dominated by a majority of low-scoring items that rarely get clicked. Following [17], we instead bin the scores of product set S into M buckets, where B_m denotes the m^{th} bucket:

$$B_m = \{P_i : \frac{m}{M} < s_i \leq \frac{m+1}{M}\} \tag{10}$$

We treat every bucket as a single example, where the score is the average score, and the true label is the average label of the examples that fall in that bucket:

$$\hat{B}(S) = \frac{\sum_{m=0}^{M-1} \frac{\sum_i \mathcal{I}(P_i \in S)\, \mathcal{I}(P_i \in B_m)\, s_i}{\sum_i \mathcal{I}(P_i \in S)\, \mathcal{I}(P_i \in B_m)}}{\sum_{m=0}^{M-1} \frac{\sum_i \mathcal{I}(P_i \in S)\, \mathcal{I}(P_i \in B_m)\, y_i}{\sum_i \mathcal{I}(P_i \in S)\, \mathcal{I}(P_i \in B_m)}}. \tag{11}$$

As for (9), we expect $\hat{B} = 1$ for a perfect model, but the bucketing ensures that products that are highly likely to convert have a larger contribution to the metric.

5 EXPERIMENTS

In this section, we describe a set of experiments designed to answer the research questions in Section 4.2.

5.1 Experiment setup

Baseline models. As discussed in Section 4.1, the proposed extra task can be easily applied on any model architecture. In this section we demonstrate it on the following baseline models.

(1) **DNN**. A vanilla multi-layer perceptron.
(2) **DeepFM** [9]. It is a popular choice to design a CTR prediction model with two parts, one responsible for capturing low order feature interactions and the other for high order feature interactions. DeepFM, consisting of a factorization machine and a deep neural network, is a widely used benchmark from this family of models.
(3) **MaskNet** [25]. Since additive operations alone are inefficient in capturing feature interactions [2], multiplicative operations are introduced in MaskNet through an instance-guided mask on feature embeddings. In previous work it outperformed popular baselines, such as DeepFM and xDeepFM [13].

Datasets.

(1) **KDD Cup 2012 track 2**[1] (referred to as "KDD2012" below) is a search advertising dataset collected from the Tencent search engine, soso.com. The original dataset comes in an aggregated form, where the same ads shown in the same session with the same contextual features are aggregated into one record. We unroll the dataset by impressions, i.e. a record with 3 impressions and 1 click is broken into 2 negative records and 1 positive record. We randomly split the provided training set into 9:1 for training and validation. We use the provided test set for testing which is held out in time. The dataset contains 2 position-related features: ad position itself, "position", and number of ad slots, "depth".
(2) **Criteo**[2] is an online advertising dataset containing ads features and click feedback. We mainly followed Liu et al. [15] for pre-processing, except for train/valid split. We randomly take 80% of "day 6–12" for training, and 20% for validation. The test set is again held out in time, taken from "day 13". Negative records are downsampled to achieve roughly 50% CTR. The feature names are anonymized, but we find some low cardinality categorical features behave like positional features. We select "C13" and "C26", where the CTRs can be 3x higher across different values, as position-like features.

Train/validation/test split. We strictly hold out the latest data for testing because a random split often hides the position bias problem. It's only when the models need to generalise to a new unseen situation that the identifiability issue becomes apparent. If the test set distribution is identical to the train set we do not observe bias, which we discovered in preliminary experiments when using a random split. Another pitfall is to validate on data with the same distribution as the test data. While it may achieve better results on the test set, it is not practical. In production systems we can only validate on historic data and hope the model can generalize well in the future. Our split is not commonly used in the literature, which makes it difficult to compare our results, but is essential to demonstrate the position bias problem.

Hyperparameter settings. We implement the bias correction methods with PyTorch [18] based on the implementations in the FuxiCTR library [28]. We follow the hyperparameter settings recommended by [9] and [25]. Most input features are already discrete, and numerical features were discretized. All of the inputs are embedded into vectors of dimension 10. For all models with a DNN part, we use 3 hidden layers with 400 neurons per layer. All activation functions are ReLUs. We use the Adam optimizer, a learning rate of 1e-4, and a dropout rate of 0.2.

Evaluation metrics. For the overall performance of the models, we look at the following two metrics: 1) LogLoss, the binary cross entropy loss. $\mathcal{L} = -y \log s - (1-y) \log(1-s)$. 2) AUC, or Area under the ROC curve, a commonly used metric for binary classification tasks. AUCs are always between 0 and 1, where higher values are better. Additionally, we look at the bias metric defined in Section 4.3 for how well a model is able to suppress bias on position-like features. We bootstrapped the test sets with sample size 100 to compute 95% confidence intervals.

[1]KDD Cup 2012 track 2: https://www.kaggle.com/c/kddcup2012-track2
[2]Criteo Display Advertising Challenge Dataset: https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/

5.2 Can we detect and quantify positional bias in existing publicly available datasets?

We trained a simple DNN baseline and the two SOTA models above on the KDD2012 and Criteo datasets. We computed the bias metric described in Section 4.3 for different combinations of position features for both the train and the test set. The results are in Figure 3.

As expected, the models look well calibrated on the train set: all the error-bars for all the models and positions overlap with one (Figures 3b, 3d). On the other hand, when looking at the test set the models don't look well calibrated at all. In particular they seem to overestimate the number of clicks at all positions for KDD2012 in Figure 3a. For Criteo (Figure 3c), the models underestimate the clicks for more prominent positions, not attributing enough value to these positions themselves.

For both datasets, the models are overestimating the expected number of clicks more for less prominent positions (bottom of the plots) (Figures 3a, 3c). This observation is compatible with our hypothesis that the model is not paying enough attention to the positional features and it's instead using other features to explain the data. On the training set the model appears able to explain the data well using other features.

However, the test set is held out in time, and when the joint distribution of the ads and positions shifts, the models don't generalize well. As a result, models tend to overestimate the click through rate on less prominent positions, indicating that the model is partially ignoring the positional features.

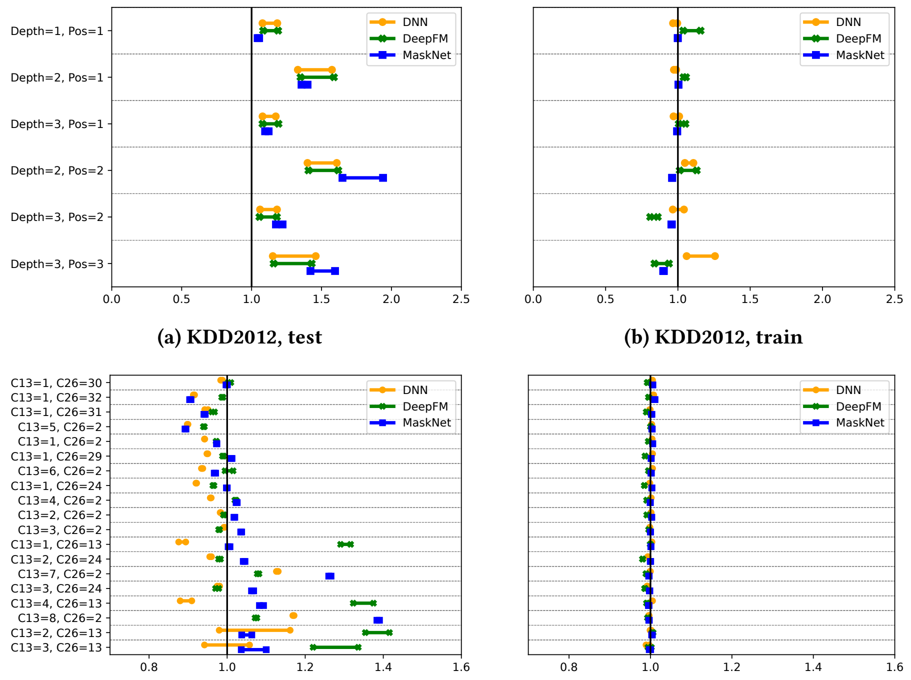
Given that our proposed metric detected position bias in all of the models we tested, we believe that it will be a useful tool for diagnosing new click-through-rate models. The pattern of results matches our hypothesis that the positional bias is due to an identifiability issue: the model can fit the training data very well while under-utilising the positional features.

5.3 Does the proposed task help in mitigating existing position bias?

To test if the extra task in Section 4.1 really helps controlling for positional bias we re-run the experiment in Section 5.2 but training the three model architectures with the additional task. We compare the extra task against the "shallow tower" approach introduced in Section 4.1 that should also control for positional bias. We report AUC and LogLoss in Table 1. We observe that the additional task does not harm the model's test performance, in fact we can see a small improvement across the board for both the metrics reported.

For KDD2012, we compare the original MaskNet against the modified versions, one with the extra task and the other with shallow tower. For the train set, all three versions of MaskNet remain calibrated in every position in Figure 4f. For the test set in Figure 4c, our approach reduces the measured bias for all combinations of position features. Our method compares favourably with the shallow tower which is better than the original MaskNet but more biased than our approach especially for Depth 2 Position 2. We repeat the analysis for the DNN baseline and DeepFM in Figures 4a, 4b and the conclusion are largely unchanged.

For Criteo, our extra task approach is effective in bringing the bias metric closer to 1 when applied on DeepFM and MaskNet, which the shallow tower fails to do. We think this is the case where



(a) KDD2012, test **(b) KDD2012, train**

(c) Criteo, test **(d) Criteo, train**

Figure 3: We report the 95% confidence interval of our bias metric for three models (DNN, DeepFM and MaskNet) stratified by positional features for the train set and the test set which is held out in time. For KDD2012 the features are sorted by position, and the most prominent positions are higher. For Criteo the features are anonymized and we can't sort from the most prominent to the least prominent, and instead we sort by observed CTR in descending order. On the train set, the models are well calibrated and don't show positional bias. On the other hand, in the test set, the models tend to underestimate click rates for positional features with high overall click rates and vice-versa. This observation indicates that the models are not using the positional features enough and are instead using some other feature to explain the data.

	KDD2012 - Test		KDD2012 - Train		Criteo - Test		Criteo - Train	
Model	AUC	LogLoss	AUC	LogLoss	AUC	LogLoss	AUC	LogLoss
DNN	<u>0.71298</u>	<u>0.14308</u>	**0.75717**.	**0.15442**	0.74756	**0.59357**	<u>0.80092</u>	<u>0.54275</u>
DNN - Shallow tower	0.71335	**0.14143**	<u>0.74625</u>	<u>0.15602</u>	<u>0.74595</u>	<u>0.59469</u>	0.80158	0.54064
DNN - Extra task	**0.71450**	0.14230	0.74946	0.15559	**0.74769**	0.59466	**0.80458**	**0.53855**
DeepFM	<u>0.70986</u>	<u>0.14327</u>	0.74158	0.15674	0.72627	<u>0.60933</u>	**0.79918**	**0.54301**
DeepFM - Shallow tower	0.71012	0.14313	<u>0.74024</u>	<u>0.15693</u>	<u>0.72523</u>	0.60926	0.79730	0.54481
DeepFM - Extra task	**0.71208**	**0.14257**	**0.74221**	**0.15665**	**0.73135**	**0.60550**	<u>0.79674</u>	<u>0.54552</u>
MaskNet	0.71275	0.14022	**0.73934**.	**0.15699**	<u>0.73960</u>	<u>0.64404</u>	0.80246	0.53872
MaskNet - Shallow tower	<u>0.71144</u>	<u>0.14105</u>	<u>0.73587</u>	<u>0.15750</u>	0.74064	0.61535	<u>0.79678</u>	<u>0.54511</u>
MaskNet - Extra task	**0.71633**	**0.13976**	0.73782	0.15720	**0.74717**	**0.59189**.	**0.80528**	**0.53570**

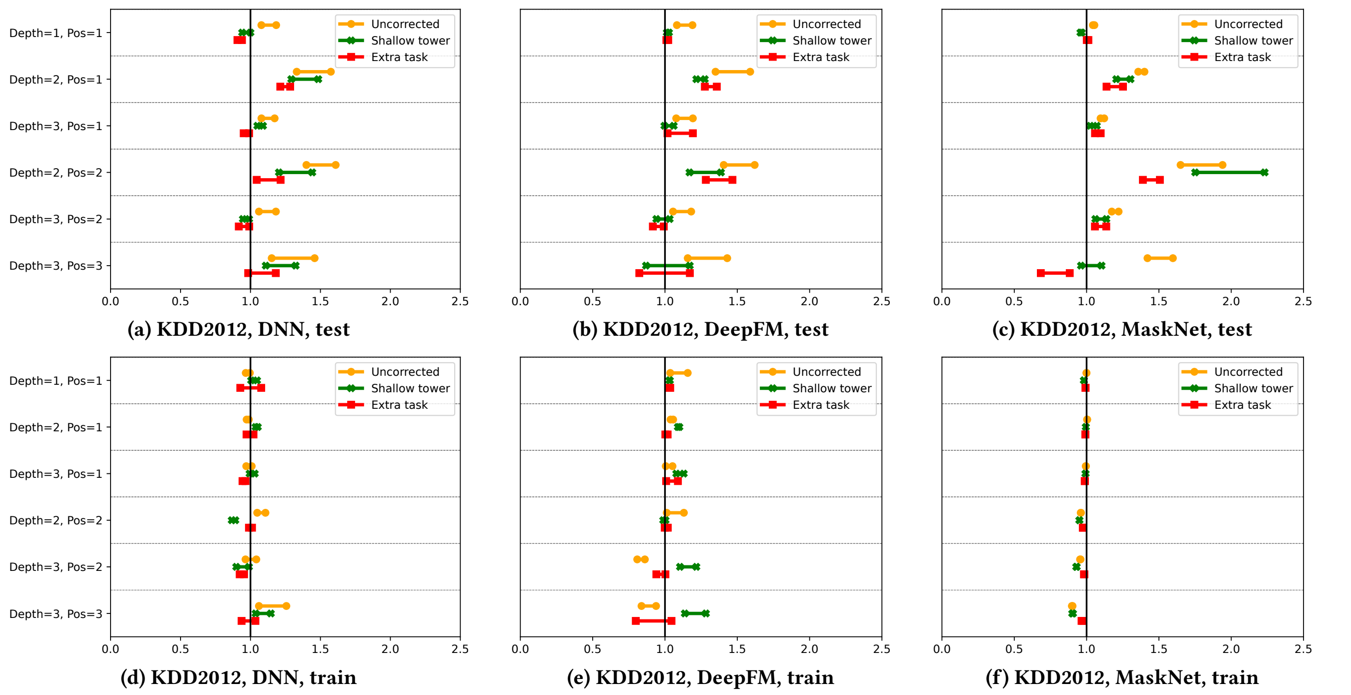
Table 1: Performance comparison for different models trained on KDD2012 and Criteo. Best results within the same model architecture are marked in bold, and worst results are underlined. Our proposed extra-task does not harm the overall model quality measured in terms of test AUC and LogLoss, in fact it's often the best performing method.

allowing the position-like features to interact with other features would help. In fact, the hypothesis is supported by the fact that shallow tower often has a worse fit as indicated by the worse AUC and LogLoss on the train set. Neither method does much for DNN.

We conclude that our proposed extra task is competitive for reducing positional bias and does not harm the overall model performance as measured with AUC or LogLoss.

5.4 Can we correct when injecting more bias?

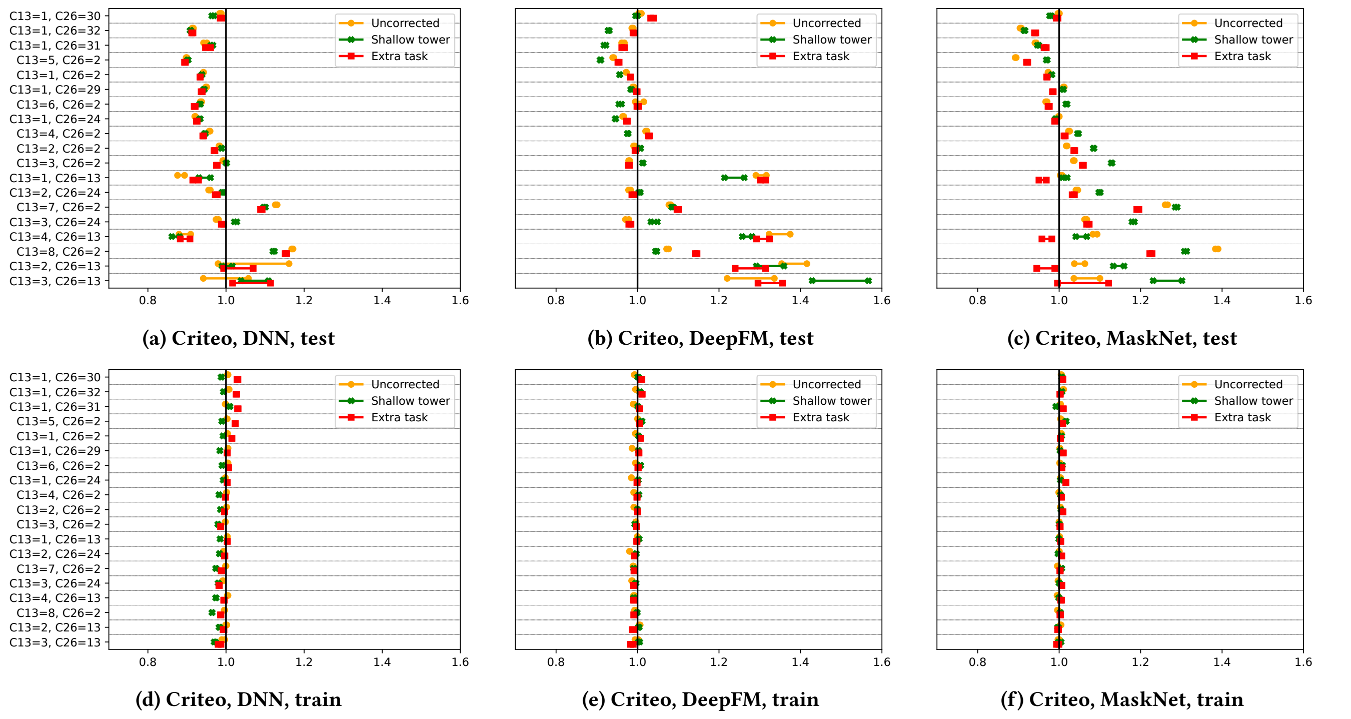
The degree of position bias exhibited in data depends a lot on the data generation and collection process [16]. Even when the bias is small and may be hard to detect, it can nevertheless be harmful in the long run when self-reinforcing. If we start with a CTR prediction model trained from scratch on mildly biased data, and do nothing specific to mitigate such bias, the model tends to



(a) KDD2012, DNN, test (b) KDD2012, DeepFM, test (c) KDD2012, MaskNet, test

(d) KDD2012, DNN, train (e) KDD2012, DeepFM, train (f) KDD2012, MaskNet, train

Figure 4: Bias metric with 95% confidence interval on train and test set for models trained with different bias mitigation methods on KDD2012. On the test set, both the shallow tower and the proposed extra-task decrease the positional bias compared to the uncorrected models. The two techniques seem to perform similarly on DeepFM, while the extra-task performs slightly better for the DNN and MaskNet.



(a) Criteo, DNN, test (b) Criteo, DeepFM, test (c) Criteo, MaskNet, test

(d) Criteo, DNN, train (e) Criteo, DeepFM, train (f) Criteo, MaskNet, train

Figure 5: Bias metric with 95% confidence interval on train and test set for models trained with different bias mitigation methods on Criteo dataset. As in Figure 4, the models are well calibrated on the train set. On the test set, both the shallow tower and the proposed extra-task decrease the positional bias, even though for the DNN they don't seem to have a large effect. For DeepFM and MaskNet the extra task seems to outperform the shallow tower bringing the points closer to 1.

Model	Bias on removed ads	AUC	LogLoss
DNN Baseline	0.853 [0.835,0.876]	0.7123	0.1429
DNN Shallow tower	0.890 [0.877,0.906]	0.7145	0.1417
DNN Extra task	**0.959 [0.943,0.976]**	**0.7186**	**0.1408**
DeepFM Baseline	0.759 [0.749,0.767]	0.7094	0.1439
DeepFM Shallow tower	**0.807 [0.796,0.817]**	0.7094	0.1433
DeepFM Extra task	0.801 [0.788,0.816]	**0.7111**	**0.1430**
MaskNet Baseline	0.736 [0.715,0.763]	0.7142	0.1407
MaskNet Shallow tower	0.736 [0.727,0.746]	0.7138	0.1408
MaskNet Extra task	**0.784 [0.769,0.801]**	**0.7154**	**0.1404**

Table 2: We report our bias metric (the mean and the 95% confidence interval in brackets) for the set of ads that are removed from the most prominent positions in the training set for KDD2012. All models tend to underestimate the click probability for this set of ads. Our proposed regularizing task does reduce bias and compares favourably with the shallow tower method in two out of three models. For DeepFM the extra-task and the shallow tower give similar results.

over-value ads previously shown on top positions. These ads will again be selected for top positions, creating a feedback loop. The bias gets amplified every time the data is used for training a new model and we will end up with increasingly more biased data as time goes. The publicly available datasets we use are collected over a relatively short amount of time (12 days and 8 days). In practice, if the training data is collected over a longer period and the model was retrained many times during that period, it is much more likely to observe a stronger bias. To simulate real long-term scenarios, we inject synthetic bias into the training set by selectively removing a subset of ads from the most prominent positions.

Since ad IDs are not available in Criteo, we perform this analysis on KDD2012 only. From the test set, we select the top 1000 ads with highest CTRs. Then we remove records with these ads at position 1 (most prominent) from the train set. The model is still able to see data points where these ads showed in other positions. We expect uncorrected models not to realize that the lower CTR of these ads in the train set is due to their low positions, and to underestimate the CTR of these ads at testing time.

We measure the bias using the metric in Section 4.3 and report our results in Table 2. While no method completely removes the negative bias for the ads removed from the top positions, our method is the one with least bias for the DNN baseline and MaskNet, and is comparable with the shallow tower for DeepFM. Moreover, our method achieves the highest AUC in all of the tested cases.

We conclude that extreme position biases are hard to completely overcome. It is easier to train on data where all ads are shown at all positions when that is possible. However, our extra task method and the shallow tower are helpful for reducing position bias, and so will more quickly identify that ads never shown in the top positions should be shown more prominently in future.

6 DISCUSSION

Position bias is a long-standing and well-known issue with click models, and it's widely appreciated that controlling for the positions observed in the train data is necessary. However, we believe that the additional problems brought by recent flexible neural models should be more widely recognized. Even with a fully-observed and simple ground-truth model, increasing model flexibility quickly leads to a position bias effect (Figure 1c). Monitoring this problem should become part of developing new neural models.

While models without position bias often generalize better, the improvement on standard performance metrics can be small (Table 1). These small differences don't reflect the large effect that position bias has on feedback effects, and fairness of the system. To focus on position bias, we propose monitoring groups of items or events using a bias-oriented metric such as (11).

We've found that it's easy to be misled about how well a model works. All our models appear to be well-calibrated on training data, so it's clear that held-out data is essential. Moreover, it was only when using test data held out in time, reflecting real world deployment, that we noticed large position bias effects. In a real-world system, data will also be driven by placements made by the model, and ideally test data should reflect that. In offline settings, simulating placement effects is enough to reveal problems with current models (Section 5.4).

Our observations connected position bias to the relatively recent observation that it is easy for flexible models to partially ignore some of their input features [19]. For common CTR prediction tasks some of the discrete features have much higher cardinality than others. It's often possible to obtain a good fit mainly using the high-cardinality features and partially ignoring the low-cardinality ones. That is, position bias can be caused by a form of overfitting, which could be mitigated by the right 'inductive bias' or regularization.

There are many possible choices of regularization, and including and tuning L2 regularization is already part of most model workflows. We focused on two position-specific inductive biases, a 'Shallow tower' that restricts how the model can use position, and an extra task that initializes the model to use position as much as possible. Both approaches improved position bias in our experiments. The correct approach depends on whether the position-like or low cardinality features need to interact with other features in the model.

None of the approaches that we discuss eliminate position bias. Biasing the model to use position more could over-correct, making the model over-estimate the CTR for items with low CTR (e.g., Figure 2c). This 'reverse' position bias would cause a system using the model to over-explore items that hadn't been shown much previously. In our experiments with real-world data we do not seem to over-correct, and there is still some position bias remaining.

It's possible that explicit regularization towards the position-only model may be useful, at the risk of a reverse bias. It's also possible that the position bias is partly caused by more complex causal effects (e.g., [7]). Our approach is potentially useful for nudging any model of any conditional distribution towards using a low-cardinality feature more. This idea could be combined with more complex causal models, and used in applications beyond CTR prediction.

ACKNOWLEDGMENTS

We would like to thank Dongge Han for providing feedback on our first draft. We also acknowledge the organizers of KDD Cup 2012 and Tencent Inc. for making the KDD Cup 2012 dataset available, and Criteo AI labs for the Criteo Display Advertising dataset.

REFERENCES

[1] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. 2018. Unbiased Learning to Rank with Unbiased Propensity Estimation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 385–394. https://doi.org/10.1145/3209978.3209986

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 46–54.

[3] Allison JB Chaney, Brandon M Stewart, and Barbara E Engelhardt. 2018. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 224–232.

[4] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2023. Bias and Debias in Recommender System: A Survey and Future Directions. *ACM Trans. Inf. Syst.* 41, 3, Article 67 (feb 2023), 39 pages. https://doi.org/10.1145/3564284

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[7] Xinyi Dai, Jianghao Lin, Weinan Zhang, Shuai Li, Weiwen Liu, Ruiming Tang, Xiuqiang He, Jianye Hao, Jun Wang, and Yong Yu. 2021. An Adversarial Imitation Click Model for Information Retrieval. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 1809–1820. https://doi.org/10.1145/3442381.3449913

[8] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (Bellevue, Washington, USA) *(ICML'11)*. Omnipress, Madison, WI, USA, 1097–1104.

[9] Huifeng Guo, Ruiming TANG, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 1725–1731. https://doi.org/10.24963/ijcai.2017/239

[10] Jiarui Jin, Yuchen Fang, Weinan Zhang, Kan Ren, Guorui Zhou, Jian Xu, Yong Yu, Jun Wang, Xiaoqiang Zhu, and Kun Gai. 2020. A Deep Recurrent Survival Model for Unbiased Ranking. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 29–38. https://doi.org/10.1145/3397271.3401073

[11] Thorsten Joachims, Laura A. Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems* 25, no 2 (2007), 7. http://doi.acm.org/10.1145/1229179.1229181

[12] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (Cambridge, United Kingdom) *(WSDM '17)*. Association for Computing Machinery, New York, NY, USA, 781–789. https://doi.org/10.1145/3018661.3018699

[13] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1754–1763.

[14] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. 2017. Model Ensemble for Click Prediction in Bing Search Ads. In *Proceedings of the 26th International Conference on World Wide Web Companion* (Perth, Australia) *(WWW '17 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 689–698. https://doi.org/10.1145/3041021.3054192

[15] Bin Liu, Niannan Xue, Huifeng Guo, Ruiming Tang, Stefanos Zafeiriou, Xiuqiang He, and Zhenguo Li. 2020. AutoGroup: Automatic Feature Grouping for Modelling Explicit High-Order Feature Interactions in CTR Prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 199–208. https://doi.org/10.1145/3397271.3401082

[16] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke. 2020. Feedback loop and bias amplification in recommender systems. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 2145–2148.

[17] Lukas Neumann, Andrew Zisserman, and Andrea Vedaldi. 2018. Relaxed Softmax: Efficient Confidence Auto-Calibration for Safe Pedestrian Detection. (2018). https://openreview.net/forum?id=S1lG7aTnqQ NIPS MLITS Workshop Machine Learning for Intelligent Transportation Systems.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron Courville, Doina Precup, and Guillaume Lajoie. 2021. Gradient Starvation: A Learning Proclivity in Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 34. 1256–1272.

[20] Zhen Qin, Suming J. Chen, Donald Metzler, Yongwoo Noh, Jingzheng Qin, and Xuanhui Wang. 2020. Attribute-Based Propensity for Unbiased Learning in Recommender Systems: Algorithm and Case Studies. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2359–2367. https://doi.org/10.1145/3394486.3403285

[21] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2008. *Dataset shift in machine learning*. Mit Press.

[22] Thomas J. Rothenberg. 1971. Identification in Parametric Models. *Econometrica* 39, 3 (1971), 577–591. http://www.jstor.org/stable/1913267

[23] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1670–1679. https://proceedings.mlr.press/v48/schnabel16.html

[24] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2818–2826.

[25] Zhiqiang Wang, Qingyun She, and Junlin Zhang. 2021. MaskNet: Introducing Feature-Wise Multiplication to CTR Ranking Models by Instance-Guided Mask. *CoRR* abs/2102.07619 (2021). arXiv:2102.07619 https://arxiv.org/abs/2102.07619

[26] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data: –A Case Study on User Response Prediction. In *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*. Springer, 45–57.

[27] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending What Video to Watch next: A Multitask Ranking System. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997

[28] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2020. Fuxictr: An open benchmark for click-through rate prediction. *arXiv preprint arXiv:2009.05794* (2020).

[29] Honglei Zhuang, Zhen Qin, Xuanhui Wang, Michael Bendersky, Xinyu Qian, Po Hu, and Dan Chary Chen. 2021. Cross-Positional Attention for Debiasing Clicks. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 788–797. https://doi.org/10.1145/3442381.3450098