Large-scale Validation of Deep Learning for Logged Bandit Feedback

Jasper Adegeest
University of Amsterdam
jasper.adegeest@student.uva.nl

Verna Dankers
University of Amsterdam
verna.dankers@student.uva.nl

Michiel van der Meer
University of Amsterdam
michiel.vandermeer@student.uva.nl

Renzo van Slooten
University of Amsterdam
renzo.vanslooten@student.uva.nl

ABSTRACT

Following recent work in the area of batch learning from logged bandit feedback, we explore the implications and performance of novel counterfactual training techniques for deep learning [9] for the task of display advertising using data from Criteo [11]. The neural network that we propose models the sparse feature input space using embeddings followed by feedforward layers. The results are compared to off-policy learning methods, amongst which are Inverse Propensity Scoring [14], Doubly Robust Optimization [5] and Policy Optimizer for Exponential Models [16]. Our results show the applicability of deep learning techniques for batch learning from bandit feedback, improving upon the Inverse Propensity Scoring method. However, we encounter overfitting in the training process and find that standard deep learning techniques have limited effect in this domain, indicating that batch learning from bandit feedback is characterized by difficulties inherent to the task that may have to be dealt with using counterfactual learning techniques to be able to fully benefit from the expressive power of deep neural networks.

1 INTRODUCTION

Many interactive systems such as search engines or display advertising algorithms collect large quantities of log data containing information about the performance of the algorithm. Typically, log entries provide information about the user's input to the system, the action taken by the system's policy and the user's corresponding feedback. Recent developments in deep learning in the field of information retrieval and beyond have given rise to a shift from modeling approaches to learning-based methods. Given the amount of data available, the problem of *counterfactual learning* has become even more apparent.

Counterfactual learning can be achieved through *Batch learning from bandit feedback* (BLBF), where logged data is used to train a new algorithm (a policy) for the interactive system at hand. The log entries are obtained using a *logging policy* that differs from the policy to be learned. Two main issues arise: Firstly, the log entries provide incomplete feedback because the user's response is only observed for the actions taken by the logging policy. Secondly, the log entries are biased because the actions preferred by the logging policy are over-represented. Incomplete feedback is called bandit feedback, and makes it distinctly different from traditional supervised learning, since there is no knowledge about the labels of unseen actions.

Whereas online learning methods are required to make actions interactively, BLBF allows us to optimize a policy without intervening with active systems. Recent work [9] has made it possible to, in addition to existing linear methods, also train deep neural networks with logged contextual bandit feedback, by creating a new deep learning training objective. In this layer, the conventional cross-entropy objective is replaced with a Counterfactual Risk Minimization (CRM) objective. Instead of calculating the exact error, an estimator of the true error is used, which allows for stochastic gradient descent (SGD) optimization.

In this paper, a deep learning approach for BLBF is applied to a dataset from Criteo concerning the task of display advertising [11]. Specifically, we focus on single slot placement, filling a single banner slot with a product advertisement which is appealing to users. The deep neural network is trained according to the CRM objective of Joachims et al. [9], is evaluated according to the evaluation methodology presented by Lefortier et al. [11] and compared to four existing approaches: a regression baseline, direct Inverse Propensity Scoring (IPS) optimization [14], Doubly Robust Optimization (DRO) [5] and optimization using the Policy Optimizer for Exponential Models (POEM) [16].

The rest of this paper is structured as follows: Section 2 elaborates on the workings of the baseline methods and recent work on deep learning for information retrieval tasks. Section 3 details our deep learning approach for the banner filling task. The experimental setup is presented in Section 4, followed by the results in Section 5. Our findings are discussed in Section 6 and lastly summarized in Section 7.

2 RELATED WORK

Originally, the bandit feedback problem was primarily encountered when evaluating placement policies. Here, feedback can be used to evaluate the effectiveness of different policies by methods such as A/B testing. However, as this is an online method of evaluation, it splits the userbase and remains very sensitive to circumstantial factors. In addition, this online method of data collection is very data inefficient and it limits the evaluation that can be performed simultaneously. Subsequently, effort was made to cast the problem into an offline evaluation task, rather than online.

Now, we consider the promising task of *off-policy evaluation* [12]: evaluating new algorithms using data obtained with the logging policy. Using historical log data allows for a more efficient way of evaluating new policies in an offline manner, in contrast to the costly online evaluation experiments. Moreover, we can exploit this offline

evaluation method by iteratively improving on models through various optimization techniques. Learning from only the feedback on performed actions, without knowing rewards for other possible actions, is counterfactual learning that underlies the BLBF problem. Counterfactual learning [4] allows us to predict the consequences of changes to a policy. Since there is only feedback on the logging policy, this theory can predict the feedback on new policies. This makes it possible to optimize policies using the predicted feedback in a loss function.

2.1 Batch Learning from Bandit Feedback

Formally, a BLBF algorithm performs the following task: A logging policy π_0 will take as input $x_i \in \mathcal{X}$, representing the features of user-action pairs taken from a candidate pool of different actions. The output is the prediction $y_i \in \mathcal{Y}$, indicating which action to perform. The loss $\delta(x_i, y_i)$[1] is a function that takes in the specific context and prediction, and returns a real number, where smaller values indicate a higher satisfaction. Now, for a new policy π_w we again need a mapping from x_i to y_i, while minimizing $\delta(x_i, y_i)$. However, due to the nature of the counterfactual learning, our data does not contain all of $\mathcal{Y}$ for all contexts $\mathcal{X}$. Additionally, some y_i are overrepresented in the data, because they were favored by π_0.

The challenges associated with BLBF are generally coped with either by reducing the problem to a supervised learning task and applying conventional learning algorithms, such as regression [10], or by using *counterfactual estimators* [4]. IPS has been introduced to construct an unbiased counterfactual estimator of policy performance using off-policy data [14]. However, IPS estimators tend to yield large variances for policies that differ greatly from the logging policy, motivating regularization approaches. One such approach that was introduced by Swaminathan and Joachims [16] is POEM, a method that includes a variance regularizer based on the CRM principle. Alternatively, hybrid models combining conventional supervised learning algorithms with counterfactual estimators were proposed, such as DRO. Below, we elaborate on the different approaches that constitute our baselines.

2.1.1 Inverse Propensity Scoring. Li et al. [12] are among the first to study an offline evaluation method for bandit algorithms that is closely related to off-policy evaluation methods from reinforcement learning literature. This evaluation method enjoys valuable theoretical guarantees including unbiasedness and resembles the propensity score estimator discussed by Rosenbaum and Rubin [14] for the evaluation of medical treatment policies. IPS [14] is used in BLBF to estimate a policy's risk based on training data (Equation 1). The IPS estimator is unbiased and has bounded variance, if the logging policy has full support. Whereas the IPS estimator is merely an approximation of a policy's performance, a learning algorithm can be put in place to maximize the estimate, which would ideally lead to a more optimal policy.

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n}\sum_{i=1}^{n}(1-\delta_i)\frac{\pi_w(y_i|x_i)}{\pi_o(y_i|x_i)} \tag{1}$$

[1]For consistency throughout the paper, $\delta(x_i,\ y_i)$ will represent a loss and $1-\delta(x_i,\ y_i)$ will represent a pay-off.

2.1.2 Self-Normalized Estimator. In the multiple efforts to employ propensity scoring for counterfactual learning, Swaminathan and Joachims [17] show that the conventional model suffers from propensity overfitting. This type of overfitting is different from the normal supervised learning overfitting, as it refers to the misuse of available data, rather than finding spurious patterns in it. Propensity overfitting is primarily caused by maximizing the sum of the new policy π_w over π_0, rather than putting a large probability mass on the samples with low loss. This effectively makes the learning process avoid all positive examples in the dataset. In turn, Swaminathan and Joachims [17] propose a self-normalized estimator, which corrects for this behavior and is referred to as self-normalized IPS estimator (SNIPS) detailed in Equation 2. Again, maximizing this estimate would ideally lead to the optimal policy.

$$\hat{R}_{SNIPS}(\pi_w) = \frac{\frac{1}{n}\sum_{i=1}^{n}(1-\delta_i)\frac{\pi_w(y_i|x_i)}{\pi_o(y_i|x_i)}}{\frac{1}{n}\sum_{i=1}^{n}\frac{\pi_w(y_i|x_i)}{\pi_o(y_i|x_i)}} \tag{2}$$

2.1.3 Doubly Robust Optimization. Dudík et al. [5] suggested to tackle the large bias of supervised methods and the large variance of IPS methods by combining them using the DRO technique. This method uses the expected reward as a baseline and corrects this if there is data available. So if at least one of the estimators is correct, unbiasedness with a lower variance can be guaranteed. To find the optimal policy the objective presented in Equation 3 ought to be maximized using a learning algorithm.

$$\hat{R}_{DRO}(\pi_w) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{((1-\delta_i)-\hat{\varrho}_a(x_i))\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)} + \hat{\varrho}_{\pi_w(x_i)}(x_i)\right] \tag{3}$$

This formula is elaborated by Dudík et al. [5]. The newly introduced $\hat{\varrho}_a(x)$ describes the estimate of the expected reward conditioned on the context and action. Note that Equation 3 replaced the indicator function $\mathbf{I}(\pi_w(x_i) = a)$ of Dudík et al. [5] with the stochastic policy $\pi_w(y_i|x_i)$, to improve comparability of $\hat{R}_{DRO}$, $\hat{R}_{IPS}$ and $\hat{R}_{SNIPS}$.

2.1.4 Counterfactual Risk Minimization. As mentioned above, IPS yields large variances for policies that differ greatly from the logging policy. This can be explained by the variance introduced by importance sampling. Swaminathan and Joachims [16] proposed adding a data-dependent regularizer to negate this introduced variance. So instead of solely optimizing for $\hat{R}_{IPS}(\pi_w)$, now an extra term is added that contains the variance term using Bernstein bounds [13]. This principle, called the Counterfactual Risk Minimization, is then used to derive the POEM learning algorithm. Decomposing the training objective using repeated variance linearization makes it possible to optimize using SGD. Without this repeated linearization, variance would be unbounded, complicating gradient optimization.

2.1.5 Other Learning Methods. Multiple other methods have been explored for the task of BLBF. Strehl et al. [15] present a method of enhancing the partial data with importance weights, based on an argmax regressor. It is a simpler variant of propensity scoring and shows corresponding performance. Furthermore, Beygelzimer and Langford [3] reduce the learning problem to a binary classification task, and show that existing fully supervised binary classification algorithms are applicable in that case.

2.2 Evaluation of Counterfactual Learning

Lefortier et al. [11] explored the possibility of having a standardized method to evaluate counterfactual learning methods. They introduce the first public dataset for BLBF. The data is acquired from Criteo and contains accurately logged propensity scores from the field of display advertising. Furthermore, a standardized evaluation method is proposed, which calculates IPS and SNIPS estimates and the standard error of the IPS estimate for each of the to be evaluated methods. They show that recent off-policy methods are able to beat supervised algorithms on the Criteo dataset. Xie et al. [20] show the effectiveness of the proposed methodology by evaluating their scoring method Maximum Likelihood Inverse Propensity Scoring with this evaluation method.

2.3 Deep Learning, Top-N recommendation and Display Advertising

Recently, Joachims et al. [9] introduced a training algorithm that allows the usage of deep neural networks for BLBF. The network is viewed as a stochastic logging policy π_w and the weights of this policy are estimated as presented in Equation 4.

$$\hat{w} = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (\delta_i - \lambda) \frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)} \tag{4}$$

The newly introduced hyperparameter λ is an unconstrained Lagrange multiplier for which the value must be selected empirically. Joachims et al. [9] use this method to train BanditNet, a variant of ResNet, and evaluate its performance on a visual object classification task using simulated logged bandit feedback. BanditNet achieved similar predictive accuracy compared to full-information training.

Although deep learning techniques for BLBF have only been introduced recently, we could highlight certain insights from related domains. For example, Alaa et al. [1] introduce a deep counterfactual network to learn effects of a medical treatment from observational data, where partial-information and selection bias also play an important role. In turn, they propose to alleviate the impact of bias through a propensity-dropout regularization scheme, that uses a different dropout probability per training example, dependent on the characteristics of the training sample.

Lastly, literature on the topics of top-N recommendation, display advertising or estimating the click-through-rate for Criteo data is related to the topic discussed in this paper.

- Top-N recommendation: Wu et al. [19] present a denoising auto-encoder for collaborative filtering that is specifically applicable for personalized top-N recommendation.
- Display advertising: Hu et al. [7] use user clustering techniques and nearest neighbour computations on products to select products to display in advertisements.
- CTR prediction for older Criteo dataset[2]: Wang et al. [18] present a novel deep learning architecture called Deep Cross Network (DCN) that consists of two parallel tracks: one regular deep network and one cross network that explicitly models feature interactions. Anil et al. [2] also propose a model for this dataset, which is a feedforward fully connected neural network using ReLu activations.

[2]https://www.kaggle.com/c/criteo-display-ad-challenge

3 APPROACH

This section describes the models implemented for the task of BLBF for display advertising. Firstly, a baseline linear model is presented that is optimized using the same objective as the neural model, but has a much simpler topology. Secondly, a neural model is introduced. The task at hand is characterized by the very large feature space due to categorical features being present in the dataset and is therefore also characterized by data sparsity. Both types of models cope with the sparse input data in a different manner. Notice that, although the network calculates only one score per input (product for the advertisement), these scores are computed for the entire candidate pool in parallel and normalized to yield probabilities.

3.1 Baseline

The first method that deals with the sparse categorical input data will be considered as the baseline model. This baseline flattens the data of one input such that each category of each feature will be treated as an individual feature. Suppose we have two features f_1 and f_2 with two categories each and one numerical feature f_3, then the unique feature-category pairs would be called $f_{1,1}$, $f_{1,2}$, $f_{2,1}$, $f_{2,2}$ and f_3. The feature vector $\mathbf{f}$ would then be represented as in Equation 5.

$$\mathbf{f} = [f_{1,1}, f_{1,2}, f_{2,1}, f_{2,2}, f_3] \tag{5}$$

If a feature was deduced from a categorical feature category, its value will be either 0 or 1, expressing membership for that specific category. To assign a score to such a feature vector, a linear transformation is applied. Finally, the output layer discussed in Section 3.2.3 is applied to compute probabilities across the candidate pool.

3.2 Network Topology

Our second approach aims to improve the baseline method by introducing a neural network approach that uses embedding layers to reduce the sparseness of the input and computes higher-order feature interactions using hidden layers and non-linear transformations. The layout of the network is visualized in Figure 1.

3.2.1 Input Layer. The input to the network is a set of product representations characterized by categorical and numerical features. Categorical features can be represented using a binary vector. However, if there is a large number of categories for a certain feature, this binary vector will be sparse and high-dimensional. To represent these vectors densely in a lower dimension, we introduce one embedding layer per categorical feature, similar to the approach of Guo et al. [6].

Assume n numerical and m categorical features. Let the i-th categorical feature f_i^{cat} have k categories and be represented as a binary vector $\mathbf{f}_i^{cat}$ of length k. This vector is mapped to an embedding by the i-th embedding layer $\mathbf{E}_i \in \mathbb{R}^{l \times k}$ to yield l-dimensional embedding $\mathbf{e}_i^{cat}$ (Equation 6). Let the i-th numerical feature f_i^{num} be represented as a scalar. This scalar is mapped to an embedding $\mathbf{e}_i^{num}$ by repeating the f_i^{num} l times. Lastly, the embeddings corresponding to numerical and categorical features are concatenated in a vector $\mathbf{h}_0$ (Equation 7).

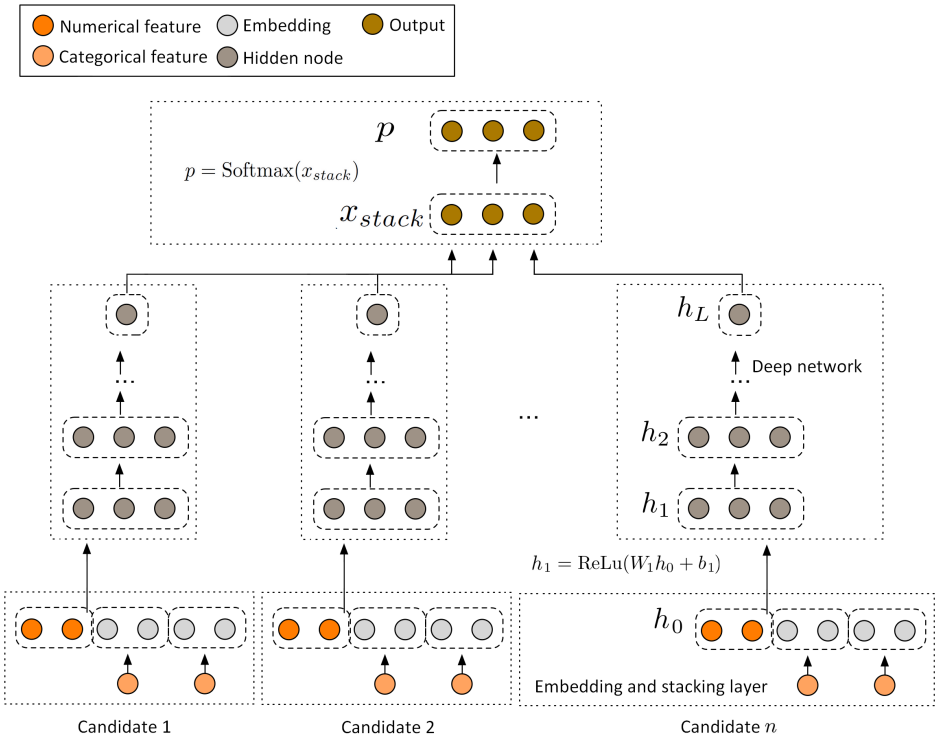


Figure 1: The topology of the neural network visualized, with the candidate pool size indicated as n.

$$\mathbf{e}_i^{cat} = \mathbf{E}_i f_i^{cat} \tag{6}$$

$$\mathbf{h}_0 = [\mathbf{e}_1^{num}; \ldots; \mathbf{e}_n^{num}; \mathbf{e}_1^{cat}; \ldots; \mathbf{e}_m^{cat}] \tag{7}$$

3.2.2 Hidden Layers. The hidden layers used are similar to the architecture of Anil et al. [2], who use a feedforward fully connected neural network in which linear layers and non-linear ReLu activations are stacked, such that layer l of the network is given by Equation 8.

$$\mathbf{h}_l = ReLu(\mathbf{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l) \tag{8}$$

3.2.3 Output Layer. In the final layer, one output number is emitted per candidate. The outputs for different products from the same candidate pool are collected and the Softmax function is applied to determine the propensities. This approach for classification allows for different numbers of products in the candidate pool. The higher the propensity, the more confident the policy is that the product should fill the banner slot.

To compute the loss, the final output layer of Joachims et al. [9] is used and the weights of the neural network are updated accordingly. The loss function is based on the constrained optimization problem showed in Equation 9. To allow for gradient optimization, Joachims et al. [9] introduce a Langrange multiplier λ to describe the loss function without constrained optimization. The loss function is given in Equation 10, and the training objective in Equation 13. Note here that the optimization problem is formulated in terms of a loss function, which results in a minimization objective. Due to the nature of the task at hand, negative examples occur much more frequently than positive examples. To control the size of the Criteo dataset, negative samples were sub-sampled by Lefortier et al. [11]. This is accounted for in this objective, by multiplying the loss by ten if we observe $\delta_i = 1$ (negative example) and dividing the summed loss by a corrected $\hat{N}$ (see Equations 10-13).

$$\hat{w} = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} \delta_i \frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)} \text{ with } \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)} = S^* \tag{9}$$

$$\mathcal{L}(w, \lambda) = \frac{1}{\hat{N}} \sum_{i=0}^{n} g(\delta_i) \cdot (\delta_i - \lambda) \cdot \frac{\pi_w(y_i \mid x_i)}{\pi_0(y_i \mid x_i)} \tag{10}$$

$$\hat{N} = \#\{\delta = 0\} + 10\#\{\delta = 1\} \tag{11}$$

$$g(\delta_i) = \begin{cases} 1, & \text{if } \delta_i = 0 \\ 10, & \delta_i = 1 \end{cases} \tag{12}$$

$$w^* = \arg\min_w \mathcal{L}(w, \lambda) \tag{13}$$

4 EXPERIMENTAL SETUP

4.1 Data

The dataset presented by Lefortier et al. [11] comes from Criteo, a leader in display advertising. The creators reduced the dataset size by aggressively sub-sampling 10% of the non-clicked samples. However, non-clicks are still highly over-represented in the dataset, as the ratio non-clicks to clicks is still 18.5. The samples representing banner ads with just one slot were used, such that the task of display

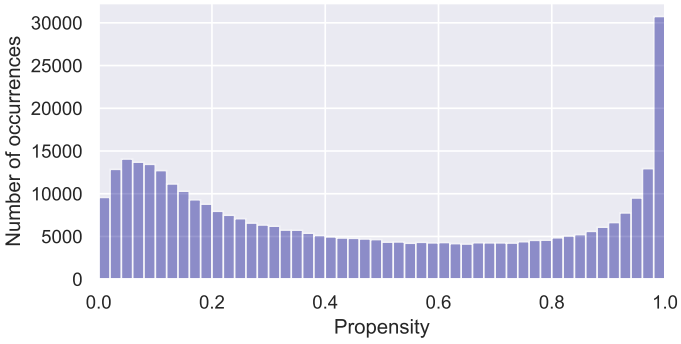


Figure 2: Distribution of propensity values for the logging policy, for products from the validation dataset.

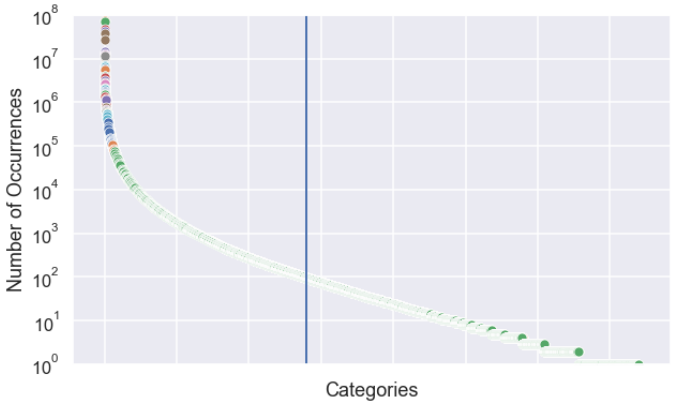


Figure 3: Number of occurrences per category visualized. The colors represent different features. The blue line indicates the categories that were removed from the training dataset.

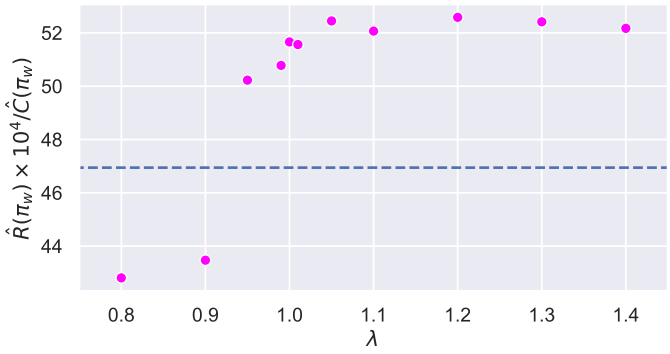


Figure 4: The effect of λ on the risk estimate for the validation dataset, for a *medium* deep neural model trained for two epochs. The estimate for the random policy is indicated in blue.

advertising is reduced to selecting one product from a candidate pool. This product should be the most appealing to the user.

The data is split into an equally sized train, test and validation set as previously done by Lefortier et al. [11]. Every data sample includes up to ten user specific features, where the first two are numerical and the remaining features are categorical. Additionally, up to 23 categorical features are presented per product. Together, they form the user-product features x_i as mentioned in Section 2. For the product that was originally selected by the logging policy, the propensity and the user's click label is provided. The products per candidate pool and the size of the pool differ per sample, with an average candidate pool size of 12. The propensity distribution of the logging policy is shown in Figure 2. Interestingly, the propensity distribution for products that received a click is very similar to the distribution for non-clicks. The average propensity for clicked products is 0.470, while the average propensity for products that were not clicked is 0.448, indicating that there is room for improvement.

The categorical features contain 84 thousand unique categories. The number of occurrences per category is visualized in Figure 3. This indicates that there are many categories for which the data is sparse. The categories appearing less than 100 times were removed to reduce the size of the feature space and to filter out categories for which an accurate representation cannot be learned due to data sparsity.

4.2 Training Setup

The baselines considered are the ones listed in the work of Lefortier et al. [11]: a random baseline, regression, IPS optimization, the DRO optimizer of Dudík et al. [5] and the POEM algorithm [16]. The best performing regression model had no Lasso regularization, a learning rate of 0.1 and an annealing learning rate schedule. The direct IPS method had no Lasso regularization, a learning rate of 1 and no learning rate schedule. The Doubly Robust method had a Lasso Regularization of 1×10^{-8}, a learning rate of 0.1, and an annealing learning rate schedule.

We experiment with different layouts for the neural models, varying the dimensionality of the embeddings, and the number of hidden layers and their dimensionality. For all experiments presented an SGD optimizer with momentum is used, similar to Joachims et al. [9], with learning rates estimated per model based on the number of trainable parameters. We apply dropout to the hidden layers of the neural network with a probability of 20%, apply Lasso regularization of 1×10^{-5} (except for *large* where 1×10^{-5} was found to be better) and train the models with a batch size of 128. The hyperparameters used for training the different models are presented in Table 1. Any deviations from these default values are mentioned with their respective models. The *medium* deep neural model was used to estimate the value of λ, as illustrated in Figure 4.

4.3 Evaluation Metrics

While training the neural models, we are *minimizing* the λ-translated loss function, analogous to $\hat{R}(\pi_w)$. For the evaluation of the baselines and neural methods, we employ the techniques presented by [11] and report performance on the following three metrics, that we aim to *maximize*:

Approach	(Hidden) Layers	Learning Rate	Size
Large	256 / 2048 / 1024 / 256	1e-03	28M
Medium	256 / 512 / 256	5e-04	11M
Small	64 / 512 / 256	3e-04	3M
Tiny	64 / 256	2e-04	2M
Sparse *Neural*	- / 32	1e-04	887K
Sparse *Linear*	- / -	3e-05	27K

Table 1: Hyperparameters for the different models evaluated. The first hidden layer size indicates the dimensionality of the embeddings used.

Approach	$\hat{R}(\pi_w) \times 10^4$	$\hat{R}(\pi_w) \times 10^4 / \hat{C}(\pi_w)$	$\hat{C}(\pi_w)$
Random	44.68 ± 2.11	45.45 ± 0.00	0.98 ± 0.02
Logger π_0	53.54 ± 0.22	53.54 ± 0.00	1.00 ± 0.00
Regression	49.34 ± 1.65	49.02 ± 0.00	1.01 ± 0.02
Direct IPS	51.04 ± 1.98	51.01 ± 0.00	1.00 ± 0.01
DRO	56.95 ± 14.02	**57.48 ± 0.01**	0.99 ± 0.01
POEM	53.18 ± 1.94	53.04 ± 0.00	1.00 ± 0.01

Table 2: Baseline performance on the Criteo testing dataset.

- $\hat{R}(\pi_w)$: The IPS estimator as presented in Equation 1, but with the sub-sampling correction applied as explained in Section 3.2.3.
- $\hat{C}(\pi_w)$: The expected importance weight, as given in Equation 14, but again with the sub-sampling correction. It is equal to the denominator in Equation 2.

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\pi_w(y_i|x_i)}{\pi_o(y_i|x_i)} \tag{14}$$

- $\hat{R}(\pi_w) \times 10^4 / \hat{C}(\pi_w)$: Similar to the SNIPS estimator, see Equation 2.

Here, it is important that the $\hat{C}(\pi_w)$ remains close to one. It serves as a control metric to identify models that potentially overfit with respect to the logging policy [17]. The most reliable performance indicator is the self-normalized estimator, as it corrects for potential propensity overfitting.

5 RESULTS

5.1 Baseline Results

The results for the baselines are shown in Table 2. Because of computational constraints, we chose to only train for five epochs.

All methods, considering they trained for significantly less epochs, obtain the expected result. Moreover, their results indicate that training for 40 epochs only slightly improves the final performance, except for the POEM algorithm.

5.2 Neural Models Results

In an effort to evaluate the correctness of our training procedure a much simpler model was implemented to reproduce a method similar to direct IPS optimization. In this type of model, we used a sparse representation of the features (as discussed in Section 3.1), and either a single low-dimensional layer, or no layer at all, to create a shallow neural model and a linear model. Intuitively, these models should perform at least as good as the direct IPS method and the shallow neural model should slightly outperform the linear model. The results for these models are presented in Table 4.

Approach	$\hat{R}(\pi_w) \times 10^4$	$\hat{R}(\pi_w) \times 10^4 / \hat{C}(\pi_w)$	$\hat{C}(\pi_w)$
Large	54.24 ± 0.00	**54.04 ± 0.00**	1.00 ± 0.01
Medium	53.47 ± 2.04	53.34 ± 0.00	1.00 ± 0.01
Small	52.08 ± 1.69	52.13 ± 0.00	0.99 ± 0.01
Tiny	52.37 ± 2.80	52.54 ± 0.00	0.99 ± 0.01

Table 3: The risk estimate on validation data after one epoch of training for deep neural networks with differing topologies.

Approach	$\hat{R}(\pi_w) \times 10^4$	$\hat{R}(\pi_w) \times 10^4 / \hat{C}(\pi_w)$	$\hat{C}(\pi_w)$
Large	54.90 ± 3.37	**54.13 ± 0.00**	1.01 ± 0.02
Sparse *Neural*	52.81 ± 1.35	52.66 ± 0.00	1.00 ± 0.01
Sparse *Linear*	51.25 ± 7.43	51.93 ± 0.00	0.99 ± 0.02

Table 4: Performance for the Criteo testing dataset for the sparse models and best-performing deep neural network according to the validation set.

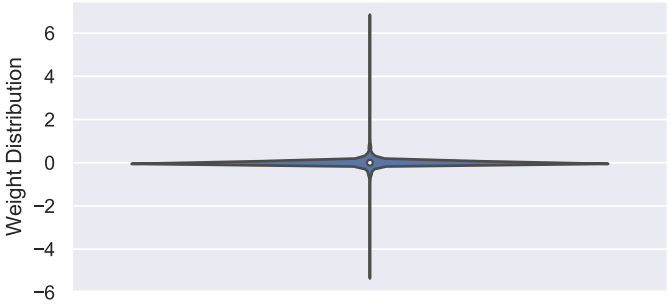


Figure 5: Weight distribution for the *sparse linear* policy.

The normalized risk estimates for validation data after one learning iteration are presented in Table 3, for which $\lambda = 1.2$ was used. The value of λ was chosen according to Figure 4.

Lastly, the best-performing neural model was selected and evaluated on the testing dataset, for which the results are shown in Table 4.

As for the logging policy, we visualized the distribution of propensity scores. Firstly, for our *medium* deep learning policy used as a stochastic policy, as can be seen in Figure 6. This represents the propensity scores that would be assigned by our model, if it were to fill the banner slots instead of the logging policy. Furthermore, Figure 7 shows the propensity scores as assigned by our model for products that were originally selected by the logging policy.

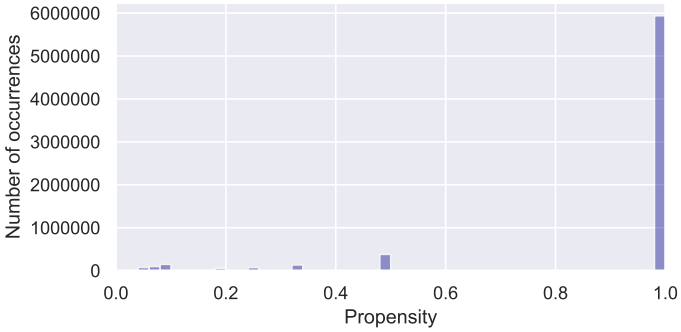


Figure 6: Distribution of propensity values for the new policy, if it were used as a stochastic policy selecting products for advertisements.

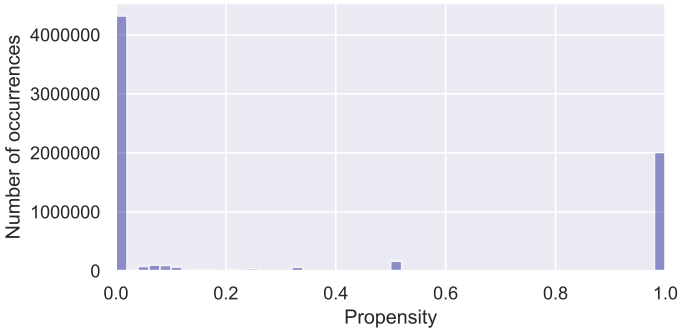


Figure 7: Distribution of propensity values for the new policy, for products from the validation set that were picked by the logging policy.

6 DISCUSSION

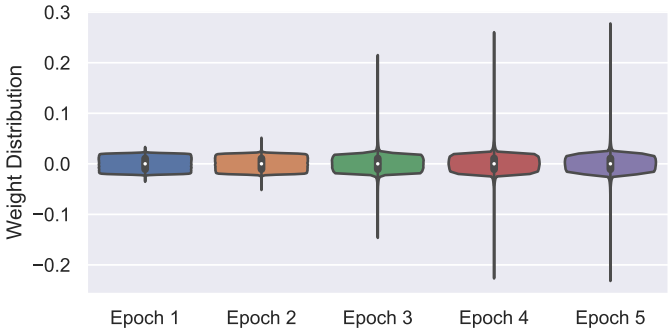
Firstly, the results for the sparse policies, presented in Table 4, illustrate the applicability of the training method and indicate that these sparse policies resemble direct IPS optimization. Additionally, even adding one layer and a non-linearity to the linear model further improves performance, indicating profitableness of deep learning techniques for the task at hand. The weight distribution of the sparse linear model is visualized in Figure 5. As this model directly maps the input features to one output, the weights can only represent importance weights of the different features. One can see that a few of the features have a particularly large influence.

Secondly, the results for the deep learning methods provide multiple valuable insights. One striking result is the influence of the value of hyperparameter λ. If the value for λ is too low, propensity overfitting occurs often. When this happens, the risk estimate for validation data becomes even lower than the estimate of the random policy. For these models the average value of the expected importance weight $\hat{C}(\pi_w)$ was 0.95, indicating propensity overfitting [17]. Models trained using $\lambda < 1$ are prone to propensity overfitting, due to the possible values of the click loss δ: $\delta = 0$ for advertisements that received a clicked and $\delta = 1$ otherwise. In the absence of a click, the translated payoff will be $\delta - \lambda \geq 0$ and the model could assign close to zero probability to the related advertisement under the policy π_w to minimize the loss. Most probability mass of policy π_w will be assigned to advertisements where $\delta - \lambda < 0$ and hence the model will overfit to the logging policy π_0.

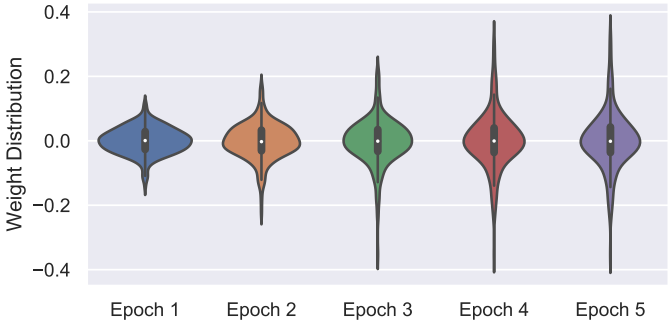
Another observation concerns the distribution of propensity values. When considering the propensity values of the neural policy for the products that were originally selected by the logging policy (see Figure 7), we clearly see that they differ greatly from the propensity values of the logging policy (see Figure 2). Whereas the logging policy has a somewhat smooth and more uniform distribution, the deep learning policy shows a very large spike at a propensity score of 1. This indicates that the neural policy is, presumably wrongly, very confident about a single product to fill a banner with, rather than having a balanced ranking of candidates. Similarly, Figure 6 illustrates the propensity values for products that are selected if we were to use our neural policy as stochastic policy to fill banners. Here, a similar pattern occurs where the neural policy is often overconfident. This can be an indicator of highly overfitting models that have learned to always favor some candidates or some specific features. Additionally, it is remarkable that both figures show a small peak around 0.5. Possibly, these cases represent products that became equal to each other because of removal of features. Further research could clarify this.

Thirdly, we want to mention that the performance of the neural models often dramatically decreased after a couple of epochs of training, due to extreme overfitting on the training dataset. To gain insight into the underlying cause, we visualized the weight distribution of one such model across five epochs of training, as shown in Figure 8. For this particular model, the decrease in performance occurred after the third epoch, where the normalized risk estimate for validation data dropped from 52.15 to 46.59, while the estimate increased for the first three epochs. The largest noticeable difference across epochs is the presence of some outliers, weights with large negative or positive values, indicating that the neural model developed a sensitivity for some specific data patterns despite the application of conventional deep learning regularization techniques and the considerable size of the dataset. This does not necessarily result in bad performance, as this already happens in the weight distribution of epoch three, albeit less extreme. It seems that fitting these features yields high risk estimates, while overfitting on those features makes the policies less robust with respect to testing data.

Lastly, although the size of the neural model, as measured by the number of trainable parameters, and its topology do affect the performance of the neural policy (Table 3), adding many more parameters yields a relatively small performance gain and larger models do not always yield higher performance. Moreover, both the linear sparse model and neural models exhibit overfitting on certain input patterns, indicating that some features listed in the Criteo training dataset have a large influence on the policies' decisions. Possibly, this concerns specialized hand-crafted features which are hard to improve upon by modeling non-linear feature interactions. Nonetheless, the neural policy did improve upon the sparse linear policy, indicating that there is something to gain by using deep neural models, provided that the overfitting problem can be combatted. Within the first five epochs of training, the neural models could improve upon the IPS baseline, but from the findings of Lefortier et al. [11] we know that training the IPS baseline for longer improves this, while we already encounter overfitting



(a) First hidden layer



(b) Output layer

Figure 8: Weight distribution for two layers of a trained *medium* deep neural network. After the third epoch a large decrease in performance occurred.

and performance decreasing in the first five epochs for the neural models. As conventional deep learning techniques did not prevent this type of overfitting, we hypothesize that specialized counterfactual learning techniques may have to be considered to combat this phenomenon.

7 CONCLUSION

We implemented a fully neural approach for BLBF for display advertising and applied this approach to the Criteo dataset presented by Lefortier et al. [11]. The network was trained using a new output layer for deep neural networks based on Inverse Propensity Scoring [9]. The dataset required a method to cope with the sparse input feature space, which resulted in a direct sparse representation, and an embedding approach. We evaluated our models according to the evaluation methodology presented by Lefortier et al. [11] and compared it to four baselines, among which is direct Inverse Propensity Scoring optimization [14], to which our approach is most comparable.

We showed that a simple sparse linear model is able to obtain similar results as presented in the literature. Furthermore, the neural approach is able to outperform direct Inverse Propensity Scoring when all models are trained for a limited number of epochs. While the baselines appear to continually improve every epoch, the neural approach immediately suffers from overfitting, providing fast but unstable learning. Although propensity overfitting seems to be solved by the loss translation hyperparameter λ of Joachims et al. [9], the neural approach still suffers from overfitting in the traditional sense, being over-confident in the propensities assigned to inputs. Effectively, it finds a small number of features which are extremely influential on the propensity values assigned. We attribute this to the nature of the task of counterfactual learning, which appears to require more sophisticated types of regularization than the conventional techniques for training deep neural networks.

7.1 Future Work

7.1.1 Variance Regularization. As discussed by Swaminathan and Joachims [16], counterfactual risk estimators can have highly fluctuating variances for different hypotheses π_w across the hypothesis space. The more similar π_w is to the logging policy π_0, the smaller the variance of the empirical risk estimate. Firstly, to dampen the effects of high variance, the importance sampling weights can be clipped to a maximum of M [8]. Secondly, an additional variance regularization term can be introduced in the training objective, that is motivated by generalization error bounds [16] (see Equation 15). That variance regularization could be beneficial for the neural policy, as could be concluded from Figure 7. Here, one can see that the distribution of propensity values of the neural policy differs highly from the distribution of the logging policy. Variance regularization could encourage our neural method to be more like the logging policy, possibly resulting in a more smooth distribution of propensity values.

$$w^* = \arg\min_w [\mathcal{L}(w, \lambda) + \gamma \sqrt{\frac{\hat{\mathrm{Var}}(\mathcal{L}(w, \lambda))}{n}}] \tag{15}$$

However, Equation 15 is not optimizable with SGD methods due to the variance term $\hat{\mathrm{Var}}(\mathcal{L}(w, \lambda))$. Joachims et al. [9] proposes a Taylor-majorization to enable SGD optimization by substituting $\hat{\mathrm{Var}}(\mathcal{L}(w, \lambda))$ as shown in Equation 16. Swaminathan and Joachims [16] describes the computation of constants A and B .

$$\hat{\mathrm{Var}}(\mathcal{L}(w, \lambda)) = \frac{1}{n} \sum_{i=1}^{n} \left[A\left(\frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}\right) + B\left(\frac{\pi_w(y_i|x_i)}{\pi_0(y_i|x_i)}\right)^2 \right] \tag{16}$$

7.1.2 Propensity Dropout. It might also be interesting to include the propensity-dropout regularization technique of Alaa et al. [1], that also aims at alleviating selection bias of π_0. By applying dropout related to the propensity, the network is thinned with a high dropout probability for samples with very high or very low propensity scores, while a low dropout is applied to the model for balanced propensity scores.

REFERENCES

[1] Ahmed M. Alaa, Michael Weisz, and Mihaela van der Schaar. 2017. Deep Counterfactual Networks with Propensity-Dropout. *CoRR* abs/1706.05966 (2017).
[2] Rohan Anil, Gabriel Pereyra, Alexandre Tachard Passos, Robert Ormandi, George Dahl, and Geoffrey Hinton. 2018. Large scale distributed neural network training through online distillation. https://openreview.net/pdf?id=rkr1UDeC-
[3] Alina Beygelzimer and John Langford. 2009. The Offset Tree for Learning with Partial Labels. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. ACM, New York, NY, USA, 129–138. https://doi.org/10.1145/1557019.1557040
[4] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013.

Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research* 14, 1 (2013), 3207–3260.

[5] 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 (ICML'11)*. Omnipress, USA, 1097–1104.

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*.

[7] Jinlong Hu, Junjie Liang, Yuezhen Kuang, and Vasant Honavar. 2018. A user similarity-based Top-N recommendation approach for mobile in-application advertising. *Expert Systems with Applications* (2018).

[8] Edward L Ionides. 2008. Truncated importance sampling. *Journal of Computational and Graphical Statistics* 17, 2 (2008), 295–311.

[9] Thorsten Joachims, Adith Swaminathan, and Maarten de Rijke. 2018. Deep Learning with Logged Bandit Feedback. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJaP_-xAb

[10] Fredrik Johansson, Uri Shalit, and David Sontag. 2016. Learning representations for counterfactual inference. In *International Conference on Machine Learning*. 3020–3029.

[11] Damien Lefortier, Adith Swaminathan, Xiaotao Gu, Thorsten Joachims, and Maarten de Rijke. 2016. Large-scale Validation of Counterfactual Learning Methods: A Test-Bed. *CoRR* abs/1612.00367 (2016).

[12] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 297–306.

[13] Andreas Maurer and Massimiliano Pontil. 2009. Empirical Bernstein Bounds and Sample-Variance Penalization. In *COLT*.

[14] Paul R Rosenbaum and Donald B Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika* 70, 1 (1983), 41–55.

[15] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. 2010. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*. 2217–2225.

[16] Adith Swaminathan and Thorsten Joachims. 2015. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*. 814–823.

[17] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*. 3231–3239.

[18] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[19] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 153–162.

[20] Yuan Xie, Boyi Liu, Qiang Liu, Zhaoran Wang, Yuan Zhou, and Jian Peng. 2018. Off-Policy Evaluation and Learning from Logged Bandit Feedback: Error Reduction via Surrogate Policy. *CoRR* abs/1808.00232 (2018).